ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδίαση και υλοποίηση πυρήνα λειτουργικού συστήματος με χρήση προσομοιωτή

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΛΑΓΟΠΟΥΛΟΥ ΑΠΟΣΤΟΛΗ

(ΑΕΜ: 1853)

Επιβλέπων : ΓΕΩΡΓΙΟΣ ΣΙΣΙΑΣ

Ιδιότητα

Καστοριά Μήνας - Έτος (παρουσίασης της εργασίας)

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδίαση και υλοποίηση πυρήνα λειτουργικού συστήματος με χρήση προσομοιωτή

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΛΑΓΟΠΟΥΛΟΥ ΑΠΟΣΤΟΛΗ

(ΑΕΜ: 1853)

Επιβλέπων : ΓΕΩΡΓΙΟΣ ΣΙΣΙΑΣ

Ιδιότητα

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ημερομηνία εξέτασης

.....	.....	.....
Ον/μο Μέλους	Ον/μο Μέλους	Ον/μο Μέλους
Ιδίοτητα Μέλους	Ιδίοτητα Μέλους	Ιδίοτητα Μέλους

Καστοριά Μήνας - Έτος(παρουσίασης της εργασίας

Περίληψη

Τα λειτουργικά συστήματα καθίστανται απαραίτητα σε ολοένα και μεγαλύτερο αριθμό συσκευών - η λίστα περιλαμβάνει από προσωπικούς υπολογιστές και έξυπνα κινητά, μέχρι έξυπνες τηλεοράσεις και ρολόγια. Στην παρούσα εργασία παρουσιάζεται η σχεδίαση και ανάπτυξη ενός πυρήνα λειτουργικού με τη χρήση προσομοιωτή. Ο προσομοιωτής μειώνει τον χρόνο που απαιτείται για φόρτωση και εκτέλεση σε σχέση με την εκτέλεση του πυρήνα σε πραγματικό υπολογιστικό σύστημα. Ο πυρήνας είναι συμβατός με την οικογένεια επεξεργαστών x86 και τους προσωπικούς υπολογιστές IBM. Στη συνέχεια γίνεται αναλυτική περιγραφή της μεθοδολογίας που ακολουθήθηκε για την ανάπτυξη του πυρήνα. Παράλληλα περιγράφονται οι βασικές αρχές του υλικού που υποστηρίζεται από τον πυρήνα. Σκοπός της εργασίας είναι μέσα από την υλοποίηση να γίνει ανάλυση των υποσυστημάτων του πυρήνα και να προσδιοριστούν οι λόγοι που καθιστούν την υλοποίηση ενός λειτουργικού συστήματος ιδιαίτερα δύσκολη.

Λέξεις Κλειδιά:

Λειτουργικό Σύστημα, πυρήνας, μονολιθικός πυρήνας, συστήματα καταμερισμού χρόνου, διεργασίες, διαχειριστής μνήμης

Abstract

Operating systems are becoming necessary for an increasing number of devices - the list includes personal computers and smartphones, as well as smart TVs and watches. This paper presents the design and development of an operating system kernel using a simulator. The simulator reduces the time required for loading and execution compared to running the kernel on a real computing system. The kernel is compatible with the x86 processor family and IBM personal computers. A detailed description of the methodology followed for the kernel's development is provided. Additionally, the basic principles of the hardware supported by the kernel are described. The purpose of this work is to analyze the kernel's subsystems and identify the reasons that make implementing an operating system particularly challenging.

KeyWords:

Operating system, kernel, monolithic kernel, time sharing systems, processes, memory manager

ΠίνακαςΠεριεχομένων

1

1. Σχεδίαση και υλοποίηση3

1.1 Μεταγλώττιση και παραγωγή αρχείου εικόνας3

1.1.1 Εργαλεία ανάπτυξης3

1.1.2 Μεταγλώττιση και εκτέλεση5

1.2 Εκκίνηση και αρχικοποίηση συστήματος8

1.2.1 Το 10

1.2.2 Ο φορτωτής εκκίνησης11

1.3 Ο πυρήνας και τα υποσυστήματα του15

1.3.1 Ο χρονοπρογραμματιστής και οι διεργασίες15

1.3.2 Ο διαχειριστής μνήμης16

1.3.3 Ο χειριστής διακοπών17

1.3.4 Ο μηχανισμός παγίδευσης18

1.3.5 Το σύστημα αρχείων19

2. Η διεπαφή χρήστη του λειτουργικού20

2.1 Προγραμματιστική διεπαφή20

2.1.1 Κλήσεις συστήματος20

2.1.2 Πρότυπη βιβλιοθήκη της C21

2.1.3 Οδηγοί συσκευών και επέκταση του πυρήνα22

2.2 Διεπαφή χρήστη22

2.2.1 Γραμμή εντολών23

2.2.2 Προγράμματα χρήστη24

2.2.3 Ένα ολοκληρωμένο παράδειγμα26

29

31

34

Λίστα Εικόνων

Εικόνα 1. Διευθυνσιοδότηση μνήμης 16-bitrealmode.13

Εικόνα 2. Η γραμμή εντολών24

Εικόνα 3. Αρχείο με κώδικα σε συμβολική γλώσσα στον κειμενογράφο26

Εικόνα 4. Το πρόγραμμα εμφανίζει τις συντεταγμένες του κέρσορα στην οθόνη.28

Λίστα Πινάκων

Πίνακας 1. Οι περιοχές μνήμης που δεσμεύει το BIOS.14

Πίνακας 2. Οι κλήσεις συστήματος του λειτουργικού21

Εισαγωγή

Σκοπός της εργασίας είναι η σχεδίαση και η ανάπτυξη ενός πυρήνα λειτουργικού συστήματος. Για τον σκοπό αυτό χρησιμοποιείται λογισμικό προσομοίωσης. Στο περιβάλλον του προσομοιωτή είναι ευκολότερο να γίνονται πολλές δοκιμές σε μικρότερο χρονικό διάστημα. Ταυτόχρονα ο προσομοιωτής κατά τον χρόνο εκτέλεσης του παρέχει πολλές πληροφορίες για το υλικό που προσομοιώνει καθιστώντας ευκολότερη την αποσφαλμάτωση. Τα παραπάνω χαρακτηριστικά τον καθιστούν κατάλληλο εργαλείο για την ανάπτυξη ενός πυρήνα λειτουργικού.

Η εργασία χωρίζεται σε δύο μεγάλα κεφάλαια. Το πρώτο κεφάλαιο περιγράφει εσωτερικά τον πυρήνα από την σκοπιά του προγραμματιστή που ασχολείται με την ανάπτυξη αυτού.Το κεφάλαιο αυτό χωρίζεται σε τρείς ενότητες.

Η πρώτη ενότητα περιγράφει τη διαδικασία για τη μεταγλώττιση και παραγωγή του αρχείου εικόνας του πυρήνα. Αρχικά περιγράφει τα εργαλεία που χρησιμοποιήθηκαν και στη συνέχεια τη διαδικασία που ακολουθήθηκε για τη μεταγλώττιση και εκτέλεση.

Η δεύτερη ενότητα περιγράφει τι συμβαίνει στον υπολογιστή κατά τη φάση της εκκίνησης. Εδώ υπάρχουν δύο υπό-ενότητες. Η πρώτη ασχολείται με το υλικολογισμικό του υπολογιστή. Η δεύτερη με το πρόγραμμα εκκίνησης που φορτώνεται στη μνήμη μόλις το υλικολογισμικό ολοκληρώσει τις εργασίες αρχικοποίησης.

Έπειτα, στην τρίτη ενότητα περιγράφεται ο πυρήνας και τα υποσυστήματα του. Γίνεται αναλυτική περιγραφή για κάθε υποσύστημα που υλοποιήθηκε. Πρώτα παρουσιάζεται ο χρονοπρογραμματιστής και ο διαχειριστής των διεργασιών. Στη συνέχεια αναλύεται ο διαχειριστής μνήμης. Έπειτα, ακολουθεί η υπό-ενότητα όπου περιγράφεται ο χειριστής διακοπών και στο τέλος του κεφαλαίου υπάρχουν δύο ακόμα υπό-ενότητες όπου παρουσιάζονται ο μηχανισμός παγίδευσης και το σύστημα αρχείων αντίστοιχα.

Στο δεύτερο κεφάλαιο αναλύεται ο πυρήνας και τα προγράμματα που εκτελούνται πάνω σε αυτόν από τη σκοπιά του τελικού χρήστη και του προγραμματιστή εφαρμογών. Αποτελείται επίσης από δύο ενότητες.

Η πρώτη ενότητα εξετάζει την εφαρμογή από τη σκοπιά του προγραμματιστή εφαρμογών. Εδώ εξετάζονται οι κλήσεις συστήματος που έχει στη διάθεση του ο προγραμματιστής εφαρμογών. Γίνεται μία συνοπτική περιγραφή όλων των κλήσεων που υποστηρίζει το λειτουργικό σύστημα. Ακολουθεί παρουσίαση της υλοποίησης της πρότυπης βιβλιοθήκης για τη συγκεκριμένη πλατφόρμα. Ο προγραμματιστής εφαρμογών έχει τη δυνατότητα να γράψει προγράμματα τα οποία χρησιμοποιούν την πρότυπη βιβλιοθήκη καθώς μέρος της έχει υλοποιηθεί. Τέλος, περιγράφεται ο τρόπος με τον οποίο το σύστημα επεκτείνει την λειτουργικότητα του με τους οδηγούς συσκευών και αναλύεται ο τρόπος με τον οποίο ένας προγραμματιστής μπορεί να γράψει τους δικούς του οδηγούς συσκευών για το παρόν λειτουργικό.

Η δεύτερη και τελευταία ενότητα του δεύτερου κεφαλαίου περιγράφει τις δυνατότητες που παρέχονται στον τελικό χρήστη. Αρχικά εξετάζεται η γραμμή εντολών. Στη συνέχεια παρουσιάζονται διάφορα βασικά προγράμματα τα οποία έχει στη διάθεση του ο χρήστης. Γίνεται μία συνοπτική περιγραφή αυτών των προγραμμάτων. Η τελευταία υπό-ενότητα παρουσιάζει ένα ολοκληρωμένο παράδειγμα χρήσης. Εξηγείται αναλυτικά η διαδικασία που πρέπει να ακολουθηθεί για να γραφεί ένα αρχείο κειμένου (που περιέχει κώδικα σε συμβολική γλώσσα) και μάλιστα στη συνέχεια περιγράφεται ο τρόπος μεταγλώττισης αυτού του αρχείου με τον συμβολομεταφραστή μέσω της γραμμής εντολών.

1. Σχεδίαση και υλοποίηση

1.1 Μεταγλώττιση και παραγωγή αρχείου εικόνας

Συνήθως σε ένα πρόγραμμα η διαδικασία της μεταγλώττισης είναι αρκετά απλή. Η μεταγλώττιση και η παραγωγή του τελικού αρχείου εικόνας (OSimage) ενός λειτουργικού συστήματος όμως είναι μία σημαντικά πιο πολύπλοκη διαδικασία. Η δομή του τελικού εκτελέσιμου που παράγεται όταν κάνουμε δόμηση (built) για ένα απλό πρόγραμμα είναι πολύ συγκεκριμένη. Είναι τέτοια ώστε να είναι απολύτως κατανοητή από το λειτουργικό πάνω στο οποίο θα εκτελεστεί. Για παράδειγμα ένα εκτελέσιμο αρχείο στα Windows (PEFormat)[1]έχει 4 bytes με τους χαρακτήρες "PE\0\0" σε ASCII κωδικοποίηση. Αμέσως μετά ακολουθούν δύο bytes των οποίων η τιμή περιγράφει τον τύπο του επεξεργαστή που αφορά το συγκεκριμένο αρχείο κ.ο.κ. Έτσι ο φορτωτής (loader)κατά τον χρόνο εκτέλεσης γνωρίζει ακριβώς πώς να φορτώσει το πρόγραμμα στη μνήμη και πώς να το εκκινήσει. Επιπροσθέτως, κατά τον χρόνο μεταγλώττισης ο μεταγλωττιστής και ο συνδέτης "γνωρίζουν" τον επεξεργαστή άλλα και το λειτουργικό πάνω στο οποίο θα εκτελεστεί η εφαρμογή. Αντίθετα, κατά την μεταγλώττιση του πυρήνα θα πρέπει να ορίσουμε τον επεξεργαστή ή την οικογένεια επεξεργαστών που είναι συμβατά με τον παραγόμενο εκτελέσιμο κώδικα. Θα πρέπει να γράψουμε ένα πρόγραμμα που θα παίζει τον ρόλο του φορτωτή και θα εκτελείται κατά την εκκίνηση (bootloader) και να συνενώσουμε τα παραπάνω δύο προγράμματα σε ένα ενιαίο δυαδικό αρχείο. Αυτό το δυαδικό αρχείο θα είναι και το τελικό αρχείο εικόνας.

1.1.1 Εργαλεία ανάπτυξης

Το μεγαλύτερο μέρος του πυρήνα του λειτουργικού συστήματος είναι γραμμένο σε C. Οι υπόλοιπες διαδικασίες που για διάφορους λόγους απαιτούσαν χειρισμούς χαμηλότερου επιπέδου γράφτηκαν σε συμβολική γλώσσα(assembly). Το ίδιο ισχύει και για τους οδηγούς συσκευών. Τα προγράμματα και οι υπηρεσίες στον χώρο του χρήστη γράφτηκαν σε C++.

Για τη μεταγλώττιση και τη σύνδεση χρησιμοποιήθηκε η έκδοση 7.0 του GNUCompilerCollection(GCC). Η έκδοση 7.0 υποστηρίζει εντολές προς τον μεταγλωττιστή (__attribute__ ((interrupt)))[2] για τη συγγραφή χειριστών διακοπτών (interrupthandlers) σε C και μας απαλλάσσει από την ανάγκη να γράψουμε αυτές τις διαδικασίες σε συμβολική γλώσσα. Προηγούμενες εκδόσεις του GCC δεν παρείχαν αυτήν τη δυνατότητα. Είναι αυτονόητο ότι, γράφοντας αυτές τις διαδικασίες σε μία γλώσσα υψηλότερου επιπέδου- όπως είναι η C σε σχέση με την assembly-, κερδίζουμε σε χρόνο υλοποίησης και φορητότητα ενώ παράλληλα, ο κώδικας μπορεί να γίνει ευκολότερα κατανοητός για όποιον θελήσει να τον μελετήσει. Γενικότερα, η έκδοση 7.0 παρέχει αρκετές χρήσιμες δυνατότητες που δεν υπάρχουν στις προηγούμενες εκδόσεις.

Άλλο ένα πλεονέκτημα του GCC είναι ότι υποστηρίζει πολλές αρχιτεκτονικές και πολλές οικογένειες επεξεργαστών[3], μειώνοντας έτσι τις αλλαγές που θα έπρεπε να γίνουν στο project σε περίπτωση που θα θέλαμε να μεταγλωττίσουμε τον πυρήνα για κάποιον άλλο επεξεργαστή. Φυσικά μία τέτοια αλλαγή θα απαιτούσε πολύ περισσότερες και σημαντικότερες αλλαγές στον πυρήνα του λειτουργικού όπως για παράδειγμα τον επανασχεδιασμό του μηχανισμού που είναι υπεύθυνος για την εναλλαγή από κατάσταση πυρήνα σε κατάσταση χρήστη και το αντίθετο ή τον επανασχεδιασμό μέρους του χρονοπρογραμματιστή -αν ο νέος επεξεργαστής απαιτούσε κάτι τέτοιο. Με λίγα λόγια, οι αλλαγές που θα έπρεπε να γίνουν για να μπορέσουμε να μεταγλωττίσουμε τον πυρήνα με άλλο μεταγλωττιστή θα ήταν μόνο ένα μικρό μέρος τον αλλαγών που θα έπρεπε να γίνουν συνολικά στο project. Ακόμα και έτσι όμως, το μεγάλο πλήθος επεξεργαστών που υποστηρίζονται από το GCC μας απαλλάσσει από την ανάγκη να αφιερώσουμε χρόνο σε περίπτωση μίας τέτοιας αλλαγής.

Μία ακόμα ισχυρή εφαρμογή που αποτελεί μέρος του GCC είναι και ο συνδέτης (ld.exe). Δίνει τη δυνατότητα να παραμετροποιήσουμε πλήρως την έξοδο που θα παράξει και ουσιαστικά να ορίσουμε τη δομή του τελικού δυαδικού αρχείου. Μπορούμε να ορίσουμε δηλαδή σε ποια θέση μέσα στο αρχείο θα τοποθετηθεί ο κώδικας, σε ποια τα δεδομένα κ.τ.λ.

Όπως αναφέρθηκε ήδη, μέρος του πυρήνα γράφτηκε σε συμβολική γλώσσα. Για τον σκοπό αυτό χρησιμοποιήθηκε ο συμβολομεταφραστής Nasm[4]. Παράγει κώδικα για την οικογένεια επεξεργαστώνx86 για 16-bit (Realmode), 32-bit (Protectedmode) και 64-bit (Longmode). Όπως θα παρουσιαστεί και στη συνέχεια, η παραγωγή κώδικα 16-bit είναι απαραίτητη διότι το πρόγραμμα εκκίνησης (bootloader) φορτώνεται από το BIOS και εκτελείται όταν ο επεξεργαστής βρίσκεται ακόμα σε κατάσταση 16-bitrealmode[5]. Το Nasm επίσης επιτρέπει μακροεντολές, γεγονός που αποδεικνύεται ιδιαίτερα χρήσιμο όταν χρειάζεται να αυτοματοποιηθεί όσο γίνεται περισσότερο η διαδικασία μεταγλώττισης και παραγωγής της δυαδικής εικόνας.

Εφόσον με τη χρήση των προαναφερθέντων εφαρμογών δημιουργηθεί ένα αρχείο με τον κώδικα και τα δεδομένα στη συνέχεια πρέπει να βρεθεί και τρόπος εκτέλεσής του. Ένας τρόπος για να το πετύχουμε αυτό θα ήταν να αντιγράψουμε με κάποιο ειδικό πρόγραμμα το αρχείο σε κάποιο μόνιμο μέσο αποθήκευσης. Δεν αρκεί να το αντιγράψουμε ως αρχείο μέσα σε κάποιο σύστημα αρχείων. Θα χρειαστεί επιπλέον να διαμορφώσουμε το μέσο αποθήκευσης και να τοποθετήσουμε το αρχείο ξεκινώντας από τον πρώτο τομέα. Στην συνέχεια θα πρέπει να επανεκκινήσουμε τον υπολογιστή επιλέγοντας αυτό το αποθηκευτικό μέσο ως μέσο εκκίνησης. Ωστόσο, η παραπάνω διαδικασία είναι χρονοβόρα, ενώ η εκτέλεση του πυρήνα κατευθείαν πάνω στο υλικό κατά τη φάση της ανάπτυξης κάνει δυσκολότερη και την αποσφαλμάτωση[6]. Αυτό συμβαίνει εν μέρει και γιατί δεν γνωρίζουμε την κατάσταση του υλικού κατά τον χρόνο

εκτέλεσης. Για παράδειγμα δεν γνωρίζουμε τι τιμές έχουν οι καταχωρητές ή το περιεχόμενο της μνήμης RAM σε μία δεδομένη χρονική στιγμή. Επιπροσθέτως, δεν μπορούμε να σταματήσουμε την εκτέλεση κατ' απαίτηση για να διεξάγουμε έλεγχους και στη συνέχεια να συνεχίσουμε την εκτέλεση όπως θα κάναμε με οποιοδήποτε άλλο πρόγραμμα θα θέλαμε να αποσφαλματώσουμε. Ένας καλύτερος και πιο ενδεδειγμένος τρόπος είναι να το εκτελέσουμε μέσω κάποιας εικονικής μηχανής ή κάποιου προσομοιωτή. Αυτό κάνει ευκολότερη τη διαδικασία της αποσφαλμάτωσης ενώ ταυτόχρονα δεν χρειάζεται να κάνουμε επανεκκίνηση κάθε φορά που θέλουμε να τρέξουμε το λειτουργικό. Για την εργασία αυτή χρησιμοποιήθηκε ο προσομοιωτής Bochs[7]. Είναι λογισμικό προσομοίωσης επεξεργαστών τις οικογένειας x86 με την ιδιότητα να προσομοιώνει επίσης πολλές βασικές συσκευές εισόδου/εξόδου. Αν και σαφέστατα πιο αργό από λογισμικά εικονικών μηχανών αποκωδικοποιεί και εκτελεί όλες τις εντολές μία προς μία.

Παρατηρούμε λοιπόν ότι η διαδικασία για τη μεταγλώττιση και εκτέλεση απαιτεί αρκετές εντολές. Για τον σκοπό αυτό δημιουργήθηκε αρχείο δέσμης ενεργειών στα Windows και ένα Linuxscript ώστε να αυτοματοποιηθεί η διαδικασία όσο το δυνατόν περισσότερο. Υπάρχει η δυνατότητα για μεταγλώττιση και εκτέλεση και από τα δύο περιβάλλοντα αρκεί φυσικά να υπάρχουν στα συστήματα αυτά όλες οι απαραίτητες εφαρμογές.

Για ανάλυση δυαδικών αρχείων χρησιμοποιήθηκε επίσης το HexEditorNeo[8]. Σε διάφορες περιπτώσεις, όπως για παράδειγμα κατά τη φάση της ανάπτυξης του συστήματος αρχείων, υπήρξε η ανάγκη να αναλυθούν δυαδικά αρχεία. Το HexEditorβοήθησε στην ανάλυση τέτοιων αρχείων.

Για ανάλυση αρχείων κώδικα σε γλώσσα μηχανής πολύ χρήσιμο αποδείχτηκε και το Objdump. Είναι αποσυμβολομεταφραστής (disassembler) και αποτελεί μέρος του GCC. Χρησιμοποιήθηκε κυρίως για λόγους αποσφαλμάτωσης (π.χ. για να ελέγξουμε ότι κατά την μεταγλώττιση ενός χειριστή διακοπής στο τέλος της διαδικασίας υπάρχει η εντολή iret και όχι η εντολή ret).

Τέλος, άλλες δύο πολύ σημαντικές εφαρμογές που χρησιμοποιήθηκαν ήταν το ImDiskToolkit[9] καθώς και το imageDisk. Η πρώτη εφαρμογή επιτρέπει τη δημιουργία εικονικών δίσκων από δυαδικά αρχεία. Η δεύτερη τη δημιουργία αντίγραφου δίσκου ως αρχείο ξεκινώντας από τον πρώτο τόμο του αποθηκευτικού μέσου, δηλαδή από τον τομέα εκκίνησης.

1.1.2 Μεταγλώττιση και εκτέλεση

Όπως αναφέρθηκε ήδη, η διαδικασία μεταγλώττισης του πυρήνα αποτελείται από αρκετά στάδια. Το πρώτο που πρέπει να γίνει είναι να μεταγλωττιστούν τα αρχεία

κώδικα σε αντικειμενικά αρχεία (objectfiles). Εδώ δεν υπάρχει κάποια ιδιαίτερη διαφορά σε σχέση με οποιαδήποτε άλλη εφαρμογή έκτος του ότι περνάμε στον μεταγλωττιστή τις παρακάτω παραμέτρους[10] μεταξύ άλλων:

-c: Ορίζουμε ότι δεν θέλουμε να γίνει αυτόματα συνένωση των αντικειμενικών αρχείων που θα παραχθούν με το τέλος της μεταγλώττισης. Διαφορετικά ο μεταγλωττιστής στο τέλος της διαδικασίας θα καλούσε αυτόματα τον συνδέτη.

-ffreestanding : Ορίζουμε πως τα παραγόμενα αρχεία δεν θα εκτελεστούν πάνω σε κάποια πλατφόρμα. Επομένως και ότι δεν υπάρχει διασύνδεση με την πρότυπη βιβλιοθήκη της C. Για παράδειγμα, δεν θα υπάρχουν οι βασικές συναρτήσεις για είσοδο/έξοδο, κάτι απόλυτα λογικό αφού η ύπαρξη τέτοιων συναρτήσεων προϋποθέτει τα εκτελέσιμα αρχεία να τρέχουν πάνω σε κάποια πλατφόρμα που παρέχει είσοδο/έξοδο. Στην συγκεκριμένη περίπτωση όμως μεταγλωττίζουμε το ίδιο το πρόγραμμα που θα παρέχει αυτήν την λειτουργία σε άλλα προγράμματα.

-ffunction-sections : Για κάθε συνάρτηση ο μεταγλωττιστής δημιουργεί διαφορετικό τμήμα κώδικα. Χωρίς αυτή την παράμετρο τυπικά ο μεταγλωττιστής θα είχε δημιουργήσει ένα τμήμα για τον κώδικα και δύο για τα δεδομένα. Θα δημιουργούσε και κάποια επιπλέον τμήματα τα οποία όμως στη συγκεκριμένη περίπτωση είναι περιττά. Ο λόγος που θέλουμε κάθε συνάρτηση σε ξεχωριστό τμήμα είναι ότι στη φάση τις σύνδεσης τον αντικειμενικών αρχείων μπορούμε να ελέγξουμε καλύτερα πως θα ενοποιηθούν στο τελικό εκτελέσιμο αρχείο όπως θα δούμε και στη συνέχεια.

-std=c99 :Ορίζουμε ως πρότυπο το ISOC99. Θα μπορούσαμε να είχαμε βάλει και κάποιο νεότερο πρότυπο της C. Το σημαντικό είναι όλα τα αρχεία που μεταγλωττίζονται να έχουν το ίδιο πρότυπο.

-m32 :Ο παραγόμενος κώδικας θα είναι για συστήματα 32-bit. Οι δείκτες και ο βασικός τύπος δεδομένων *int* θα καταλαμβάνουν 4 bytes. Κάτι επίσης απόλυτα λογικό αφού το σύστημα που αναπτύσσουμε αφορά περιβάλλον 32-bit.

-mgeneral-regs-only : Αυτή η παράμετρος χρειάζεται μόνο στη μεταγλώττιση χειριστών διακοπών. Ο παραγόμενος κώδικας χρησιμοποιεί μόνο τους καταχωρητές γενικού σκοπού. Αν δεν ορίσουμε αυτή την παράμετρο σε κάποιο αρχείο που περιέχει χειριστή διακοπτών και κάποιος καταχωρητής πέραν τον καταχωρητών γενικού σκοπού χρησιμοποιηθεί, τα δεδομένα που περιέχει πριν και μετά από μία διακοπή δεν θα είναι τα ίδια όπως και θα έπρεπε.

Στην συνέχεια θα πρέπει να μεταγλωττίσουμε και τα αρχεία που περιέχουν κώδικα συμβολικής γλώσσας. Οι μόνες παράμετροι που θα πρέπει να περάσουμε στο NASM είναι το αρχείο εισόδου και το όνομα του παραγόμενου αρχείου. Θα πρέπει επίσης να ορίσουμε και το format του παραγόμενου αρχείου με την παράμετρο –f. Δεν παίζει ιδιαίτερο ρόλο η μορφή του παραγόμενου αρχείου αρκεί ο συνδέτης στη συνέχεια να

είναι σε θέσει να συνενώσει όλα τα παραγόμενα αρχεία. Η μορφή που επιλέχτηκε για τον σκοπό αυτό είναι το Executable and Linkable Format (elf).

Το επόμενο βήμα είναι η συνένωση όλων τον αντικειμενικών αρχείων. Για τον σκοπό αυτό εκτελούμε το ld.exe με παράμετρο -m i386pe η οποία ορίζει την αρχιτεκτονική του παραγόμενου αρχείου. Και πάλι δεν ενδιαφέρει ιδιαίτερα η αρχιτεκτονική. Άλλη μία σημαντική παράμετρος είναι το -Ttext 0x500. Έτσι ορίζουμε τη θέση μνήμης όπου και θα φορτωθεί ο κώδικας κατά τη φόρτωση του προγράμματος από το πρόγραμμα εκκίνησης. Τέλος, με την παράμετρο –T δηλώνουμε ένα αρχείο εισόδου που παραμετροποιεί περαιτέρω την έξοδο. Το αρχείο εισόδου αυτό είναι στην ουσία ένα script που περιέχει τη σειρά με την οποία θέλουμε να καταχωρηθούν οι συναρτήσεις στο τμήμα του κώδικα του εκτελέσιμου. Ορίζουμε λοιπόν ότι η συνάρτηση _main που είναι και το σημείο εισόδου του προγράμματος θα βρίσκεται πρώτη και έπειτα θα ακολουθούν όλες οι υπόλοιπες συναρτήσεις. Ορίζουμε επίσης ότι το τμήμα του κώδικα θα ξεκινάει από τη θέση μνήμης 0x500. Έτσι, το πρόγραμμα εκκίνησης όταν θα φορτώσει τον πυρήνα στη μνήμη και θα ολοκληρώσει όλες τις διαδικασίες αρχικοποίησης γνωρίζει ότι θα πρέπει να μεταφέρει την εκτέλεση στη θέση 0x500.

Μετά και από την σύνδεση των αντικειμενικών αρχείων έχουμε ένα εκτελέσιμο που όμως έχει λάθος δομή. Το GCCακόμα και στην πιο πρόσφατη έκδοση στα Windows δεν υποστηρίζει απ’ ευθείας από τον συνδέτη κάποια παράμετρο ώστε να παράξουμε το δυαδικό αρχείο που θέλουμε[11]. Αντίθετα, το εκτελέσιμο που παράγει έχει formatPE[1].Δηλαδή αυτό που θα είχε ένα εκτελέσιμο στα Windows. Αν η ανάπτυξη της εφαρμογής γίνεται σε Linux περιβάλλον με GCC μπορούμε με μία παράμετρο στον συνδέτη να παράξουμε άμεσα το σωστό αρχείο. Η παράμετρος αυτή είναι η --*oformatbinary.* Ωστόσο επειδή θέλουμε η εφαρμογή να μπορεί να αναπτυχθεί και από περιβάλλον Linux αλλά και από Windows παράγουμε το εκτελέσιμο σε PEformat και έπειτα με την εφαρμογή *objcopy* που αποτελεί μέρος του GCC μετατρ'επουμε στο επιθυμητό format.

Αφού πλέον έχουμε το αρχείο που περιέχει όλα τα δεδομένα και τον κώδικα πρέπει να μεταγλωττίσουμε και τον φορτωτή εκκίνησης. Ο φορτωτής εκκίνησης όμως πρέπει να γνωρίζει κάθε φορά το μέγεθος του πυρήνα ώστε να ξέρει πόσους τομείς πρέπει να μεταφέρει από το αποθηκευτικό μέσο στη μνήμη του υπολογιστή. Καθώς προσθέτουμε νέες λειτουργίες στον πυρήνα ο κώδικας και τα δεδομένα θα είναι περισσότερα με αποτέλεσμα και το αρχείο που θα πρέπει να φορτωθεί να είναι μεγαλύτερο. Αυτό πρακτικά θα σήμαινε ότι κάθε φορά που κάνουμε μεταγλώττιση πριν τρέξουμε την προσομοίωση, θα έπρεπε να κοιτάμε πόσα bytes έχει το αρχείο, να διαιρούμε με το 512 για να υπολογίσουμε πόσους τομείς πρέπει να φορτώσουμε και να επεξεργαζόμαστε το αρχείο του φορτωτή εκκίνησης βάζοντας τον αριθμό που υπολογίσαμε. Για να αυτοματοποιηθεί η παραπάνω διαδικασία πρέπει πρώτα μέσα στο αρχείο δέσμης ενεργειών και στο αρχείο κελύφους των Linux να γράψουμε μία διαδικασία που θα υπολογίζει από πόσους τομείς αποτελείται το αρχείο. Έπειτα όταν μεταγλωττίζουμε τον

φορτωτή εκκίνησης περνάμε ως παράμετρο αυτόν τον αριθμό. Από την άλλη πλευρά, στον φορτωτή εκκίνησης έχουμε ορίσει τον αριθμό των τομέων που θα διαβάσει από το αποθηκευτικό μέσο ως μακροεντολή. Ο συμβολομεταφραστής NASM που χρησιμοποιούμε υποστηρίζει μακροεντολές. Στην ουσία πρόκειται για μία παρόμοια λειτουργιά με την ντιρεκτίβα*#define*της C.

Μόλις ολοκληρωθεί και η μεταγλώττιση του φορτωτή εκκίνησης πρέπει να συνενώσουμε τα δύο αρχεία (τον φορτωτή εκκίνησης και τον πυρήνα του λειτουργικού) σε ένα αρχείο. Πρώτα πρέπει να ξεκινάει το αρχείο του φορτωτή εκκίνησης και έπειτα να ακολουθεί ο πυρήνας. Αυτό είναι σημαντικό καθώς ο φορτωτής εκκίνησης θα πρέπει να βρίσκεται στον πρώτο τόμο. Με τον όρο συνένωση εδώ εννοούμε απλά ότι στο παραγόμενο αρχείο ακριβώς μετά το τελευταίο byte του πρώτου από τα δύο αρχεία που συνενώνουμε ακολουθεί το πρώτο bytetου δεύτερου αρχείου. Ακριβώς δηλαδή όπως θα συνενώναμε και δύο αλφαριθμητικά. Και σταWindows και στα Linux η παραπάνω διαδικασία γίνεται εύκολα στην γραμμή εντολών. Έχοντας τώρα το τελικό αρχείο εικόνας του λειτουργικού απλά καλούμε τον προσομοιωτή. Αν θέλαμε να εκτελέσουμε το λειτουργικό σε πραγματικό υπολογιστή και όχι σε προσομοιωτή θα έπρεπε να κάνουμε και ένα επιπλέον βήμα. Να γράψουμε την εικόνα σε κάποιο αποθηκευτικό μέσο φροντίζοντας ότι ο φορτωτής εκκίνησης θα βρίσκεται στον πρώτο-πρώτο τόμο του αποθηκευτικού μέσου και να εκκινήσουμε τον υπολογιστή από αυτό το αποθηκευτικό μέσο.

Τα αρχεία *execute.bat* και *execute.sh* περιέχουν όλες τις εντολές και εκτελώντας τα εφόσον υπάρχουν όλες οι εφαρμογές, μεταγλωττίζουμε και εκκινούμε την προσομοίωση.

1.2 Εκκίνηση και αρχικοποίηση συστήματος

Κατά την εκκίνηση του υπολογιστή πρέπει να φορτωθεί ο πυρήνας στη μνήμη αλλά και να γίνει αρχικοποίηση σε όλα τα υποσύστηματά του (π.χ. διαχειριστής μνήμης). Πρέπει επίσης να αρχικοποιηθούν και όλες οι συσκευές, συμπεριλαμβανομένου και του επεξεργαστή. Αξίζει να σημειωθεί ότι η πολυπλοκότητα και το πλήθος τον ενεργειών που πρέπει να γίνουν εξαρτάται από την πλατφόρμα[12]. Η οικογένεια επεξεργαστών x86 λόγω της συμβατότητας που διατηρεί με επεξεργαστές τις ίδιας οικογένειας παλαιότερων γενεών αλλά και λόγω του πλήθους τον δυνατοτήτων που υποστηρίζει κάνει την αρχικοποίηση του κάπως πιο πολύπλοκη από τους περισσότερους επεξεργαστές. Παράλληλα, το ίδιο το λειτουργικό σύστημα διαθέτει πολλά υποσυστήματα που χρήζουν αρχικοποίησης κατά την εκκίνηση.

Μέχρι στιγμής δεν έχει γίνει κάποιος διαχωρισμός στα στάδια της εκκίνησης. Τυπικά μπορούμε να πούμε ότι σε ένα σύστημα PC έχουμε δύο στάδια κατά την εκκίνηση. Πρώτα το firmware του υπολογιστή (BIOS ή UEFI) κάνει διάφορους ελέγχους πολύ χαμηλού επιπέδου (για παράδειγμα ελέγχει αν υπάρχει εγκατεστημένη RAM στον

υπολογιστή και αν λειτουργεί κανονικά). Και ύστερα ο φορτωτής εκκίνησης αναλαμβάνει να διαμορφώσει κατάλληλα το περιβάλλον και να φορτώσει το λειτουργικό σύστημα.

Το firmware λοιπόν ελέγχει το διαθέσιμο υλικό στον υπολογιστή, κάνει τις απαραίτητες αρχικοποιήσεις για κάθε συσκευή και παρέχει βασικές διαδικασίες για τα προγράμματα που θα εκτελεστούν μετά από αυτό. Οι διαδικασίες αυτές αφορούν κυρίως είσοδο και έξοδο αλλά δεν περιορίζονται μόνο εκεί. Για παράδειγμα το BIOS υποστηρίζει μία διαδικασία που επιστρέφει τη διαθέσιμη φυσική μνήμης που είναι εγκατεστημένη στο σύστημα. Η ίδια διαδικασία επιστρέφει επίσης έναν χάρτη μνήμης με όλες τις περιοχές μνήμης που είναι διαθέσιμες και όλες τις περιοχές που για διάφορους λόγους δεν είναι διαθέσιμες (π.χ. χρησιμοποιούνται από το ίδιο το BIOS) . Έπειτα το firmware φορτώνει στη μνήμη τα πρώτα 512 bytes από τον τομέα εκκίνησης καθώς εκεί πρέπει να βρίσκεται ο φορτωτής εκκίνησης (bootloader). Τέλος, συνεχίζει τη ροή εντολών από αυτό το πρόγραμμα που μόλις φόρτωσε στη μνήμη.

Προηγουμένως αναφέραμε ότι ο φορτωτής εκκίνησης αναλαμβάνει να διαμορφώσει κατάλληλα το περιβάλλον και να φορτώσει το λειτουργικό σύστημα. Η κατάλληλη διαμόρφωση του περιβάλλοντος συνίσταται στο να παραμετροποιήσει με τέτοιο τρόπο το σύστημα ώστε να είναι σε θέση να εκτελεστεί το λειτουργικό και ακόμα να μεταβιβάσει στο λειτουργικό τις πληροφορίες που το ίδιο το λειτουργικό δεν θα είναι σε θέση να ανακτήσει αργότερα. Για παράδειγμα, αν το πρόγραμμα εκκίνησης δεν θέσει τον επεξεργαστή σε λειτουργία protectedmode ο επεξεργαστής δεν μπορεί να εκτελέσει εντολές 32-bit και άρα δεν μπορεί να εκτελέσει τον ίδιο τον πυρήνα. Επίσης μετά τη μετάβαση από 16-bitrealmode όπου βρίσκεται εκείνη τη στιγμή ο επεξεργαστής, σε κατάσταση 32-bitprotectedmode, δεν θα υπάρχει κάποιος τρόπος να μάθουμε την ποσότητα της εγκατεστημένης Ram στον υπολογιστή καθώς ο μόνος αξιόπιστος τρόπος να μάθουμε αυτήν την πληροφορία είναι με κάποια κλήση στο BIOS που όμως μπορεί να εκτελεστεί μόνο σε 16-bitrealmode.

Αξίζει να σημειώσουμε ότι σε πολλές υλοποιήσεις ο ίδιος ο φορτωτής εκκίνησης έχει περισσότερα από ένα στάδια, συνήθως δύο. Δηλαδή ένα πρόγραμμα που με τη σειρά του φορτώνει ένα άλλο πρόγραμμα που με τη σειρά του φορτώνει τον πυρήνα. Στην παρούσα εργασία δεν υπήρξε ανάγκη για κάτι τέτοιο. Κάτι ακόμα που είναι άξιο αναφοράς είναι πως δεν είναι απαραίτητο ένα λειτουργικό σύστημα να φορτώνεται αποκλειστικά από έναν φορτωτή εκκίνησης. Για παράδειγμα ο φορτωτής εκκίνησης GRUB μπορεί να χρησιμοποιηθεί και για το Linux και για τα Windows.Τα Linux επίσης μπορούν να χρησιμοποιήσουν σαν φορτωτή εκκίνησης το GRUB ή το LILO μεταξύ άλλων.

Μόλις τελειώσει και ο φορτωτής εκκίνησης το έργο του, ο έλεγχος μεταβιβάζεται επιτέλους στον πυρήνα. Και ο πυρήνας με την σειρά του κατά την φόρτωση και εκτέλεση του, θα χρειαστεί να αρχικοποιήσει όλα τα υποσυστήματα του (π.χ. χρονοπρογραμματιστή, διαχειριστή διεργασιών κ.ο.κ.). Η εξήγηση της αρχικοποίησης

του κάθε υποσυστήματος θα γίνεται στις αντίστοιχες ενότητες που περιγράφουν το κάθε υποσύστημα και όχι σε αυτήν την ενότητα.

1.2.1 Το υλικολογισμικό

Πριν αναλύσουμε τον φορτωτή εκκίνησης καλό θα ήταν να εξετάσουμε σε τι κατάσταση αφήνει το υλικολογισμικό (firmware) το υλικό του υπολογιστή. Στα PC αρχικά υπάρχουν δύο διαφορετικά firmware: το παλαιότερο BIOS και το νεότερο UEFI. Το κάθε ένα από αυτά χρειάζεται και διαφορετικό φορτωτή εκκίνησης. Ωστόσο οι υλοποιήσεις του νεότερου UEFI τυπικά έχουν επιλογή για προς τα πίσω συμβατότητα με το BIOS. Στον πυρήνα δεν απαιτείται κάποια αλλαγή ανεξάρτητα από το υλικολογισμικό του υπολογιστή. Στην παρούσα υλοποίηση ο φορτωτής εκκίνησης είναι συμβατός με το BIOS. Αν θέλουμε να εκτελεστεί ο πυρήνας σε υπολογιστή με UEFI θα πρέπει να ενεργοποιήσουμε πρώτα μέσα από τις ρυθμίσεις του UEFI την προς τα πίσω συμβατότητα με το BIOS. Το όνομα αυτής της παραμέτρου μπορεί να είναι λίγο διαφορετικό ανά κατασκευαστή και έκδοση. Μία χρήσιμη λειτουργία που θα μπορούσε να προστεθεί σε μελλοντική έκδοση θα ήταν ένας φορτωτής εκκίνησης για το UEFI.

Όσον αφορά το BIOS ελάχιστα πράγματα είναι προτυποποιημένα [13]. Αυτό σημάνει ότι το διαφορετικοί κατασκευαστές και διαφορετικές εκδόσεις του BIOS μπορεί να αφήνουν το σύστημα σε εντελώς διαφορετική κατάσταση από ότι θα περιμέναμε. Αυτό κάνει τον φορτωτή εκκίνησης σημαντικά πιο πολύπλοκο πρόγραμμα. Συγκεκριμένα γνωρίζουμε με βεβαιότητα τα εξής:

1. Ο φορτωτής εκκίνησης βρίσκεται στην φυσική διεύθυνση 0x7C00.
2. Ο επεξεργαστής βρίσκεται σε κατάσταση 16-bitrealmode.
3. Ο καταχωρητής DL περιέχει τον αριθμό του δίσκου στον οποίο βρίσκεται και ο φορτωτής εκκίνησης.
4. Μόνο τα πρώτα 512 bytes(ο πρώτος τομέας δηλαδή) έχει μεταφερθεί στη φυσική μνήμη.

Γνωρίζουμε επίσης ότι για να ξεκινήσει η εκτέλεση του προγράμματος εκκίνησης πρέπει τα τελευταία δύο bytes του πρώτου τομέα να έχουν τις τιμές 0x55 και 0xAA αντίστοιχα. Αν δεν έχουν αυτές τις τιμές, το BIOS θα θεωρήσει ότι δεν πρόκειται για σωστά διαμορφωμένο τόμο εκκίνησης και δεν μεταφέρει ποτέ τον έλεγχο σε αυτόν.

Όπως αναφέραμε και νωρίτερα το BIOS εκτός των άλλων παρέχει και ένα σύνολο από διαδικασίες για είσοδο/έξοδο αλλά και για διάφορες άλλες λειτουργίες. Για περισσότερες πληροφορίες όσον αφορά τις διαθέσιμες κλήσεις του BIOS (BIOSInterruptCalls) μπορεί κανείς να δει τα [14]Δίνουν μία πλήρη λίστα με όλες τις διαθέσιμες κλήσεις.

Για οποιαδήποτε άλλη παράμετρο δεν μπορούμε να ήμαστε σίγουροι για την κατάσταση στην οποία βρίσκεται. Η στρατηγική που ακολουθούμε σε αυτές τις

περιπτώσεις είναι να ελέγχουμε πρώτα την κατάσταση στην οποία βρίσκεται εκείνη τη στιγμή το σύστημα και ανάλογα αν υπάρχει ανάγκη να κάνουμε την κατάλληλη αλλαγή.

1.2.2 Ο φορτωτής εκκίνησης

Αφού το BIOS έχει ολοκληρώσει όλες τις εργασίες, δίνει τον έλεγχο στον φορτωτή εκκίνησης. Ο κώδικας του προγράμματος αυτού στην παρούσα εργασία βρίσκεται στο αρχείο *protected.asm.*

Η πρώτη ενέργεια που γίνεται, είναι να ελέγξουμε εάν η γραμμή A20 (A20 Line) είναι ενεργοποιημένη. Πρόκειται ουσιαστικά για το 21^{ο}bit του δίαυλου διευθύνσεων. Εάν δεν είναι ενεργοποιημένη αυτή η γραμμή τότε δεν έχουμε πρόσβαση ούτε για ανάγνωση/εγγραφή ούτε και για εκτέλεση σε όλες τις διευθύνσεις όπου το bit20 έχει τιμή ίση με ένα όταν μεταβούμε σε Protectedmode. Δηλαδή όλες τις διευθύνσεις μεταξύ του 2^{ου} και 3^{ου}MB, του 4^{ου} και το 5^{ου}, του 6^{ου} και του 7^{ου}κ.ο.κ. Για παράδειγμα αν με κάποια εντολή προσπαθήσουμε να αποθηκεύσουμε έναν οποιοδήποτε αριθμό στη διεύθυνση 0x100001 (10000000000000000001 σε δυαδική μορφή) χωρίς να είναι ενεργοποιημένη η γραμμή A20 τελικά θα καταλήξουμε να αποθηκεύσουμε τον αριθμό αυτό στην διεύθυνση 0x1 (00000000000000000001 σε δυαδική μορφή) αφού το bit 20 θα πάρει μηδενική τιμή πάρα το γεγονός ότι θα έπρεπε να πάρει την τιμή ένα.Σε κάποιες υλοποιήσεις λοιπόν το BIOS μπορεί να έχει ενεργοποιήσει από μόνο του τη γραμμή A20 σε άλλες υλοποιήσεις όμως όχι. Οι λόγοι για τους οποίους η γραμμή A20 μπορεί να ενεργοποιείται/απενεργοποιείται στους σημερινούς επεξεργαστές είναι καθαρά για προς τα πίσω συμβατότητα με παλαιότερα μοντέλα επεξεργαστών τις οικογένειας x86. Για περισσότερες πληροφορίες μπορείτε να δείτε εδώ[15].

Συνοπτικά αυτό που μπορούμε να πούμε είναι ότι οι επεξεργαστές της εποχής υποστήριζαν διευθύνσεις μνήμης μέχρι και 1 MB. Ωστόσο χρησιμοποιούσαν ζεύγη καταχωρητών (segment, offset) οι οποίοι μπορούσαν να παραγάγουν διευθύνσεις μεγαλύτερες του 1 MB. Αυτό που συνέβαινε όταν κάποιος ήθελε να προσπελάσει μία διεύθυνση μεγαλύτερη του 1 MB είναι ότι το 21^{ο}bit ήταν πάντα 0 με αποτέλεσμα να έχει τελικά πρόσβαση στην διεύθυνση που είναι ακριβώς ένα MB μικρότερη. Κάποια λογισμικά που ήθελαν να έχουν άμεση πρόσβαση στις χαμηλές διευθύνσειςκυρίως για άμεση πρόσβαση στα γραφικά (και να αποφεύγουν να χρησιμοποιούν τις πιο αργές κλήσεις τουBIOSγια την ίδια δουλειά) χωρίς να χρειαστεί να αλλάζουν τους καταχωρητές τμήματος κάθε φορά, εκμεταλλεύτηκαν αυτή την ιδιότητα των επεξεργαστών. Τα νέα μοντέλα της ίδιας οικογένειας υποστήριζαν στο μέλλον μεγαλύτερο εύρος διευθύνσεων και αυτά τα προγράμματα δεν λειτουργούσαν πλέον. Για να είναι σε θέση να εκτελούν αυτά τα προγράμματα οι νέοι επεξεργαστές προστέθηκε η δυνατότητα ενεργοποίησης/απενεργοποίησης της γραμμής A20.

Ο πιο απλός τρόπος για να ελέγξουμε αν η γραμμή A20 είναι ενεργοποιημένη ή όχι είναι να αποθηκεύσουμε έναν τυχαίοαριθμό σε κάποια διεύθυνση μεγαλύτερη από 1

MB και μικρότερη του 2 MB. Στη συνέχεια αποθηκεύουμε έναν άλλο αριθμό στην αντίστοιχη διεύθυνση που είναι μικρότερη του 1 MB. Για να υπολογίσουμε αυτή τη διεύθυνση αλλάξουμε το 21^{o}bit της αρχικής διεύθυνσης από 1 σε 0. Έπειτα ελέγχουμε αν η αρχική διεύθυνση έχει την τιμή που είχαμε αποθηκεύσει εξ αρχής ή αν έχει πάρει την τιμή που αποθηκεύσαμε στη δεύτερη διεύθυνση. Αν κράτησε την αρχική τιμή η γραμμή A20 είναι ενεργοποιημένη και δεν χρειάζεται να κάνουμε κάτι επιπλέον. Αν όχι, -το πιθανότερο- θα πρέπει να ενεργοποιήσουμε τη γραμμή A20.

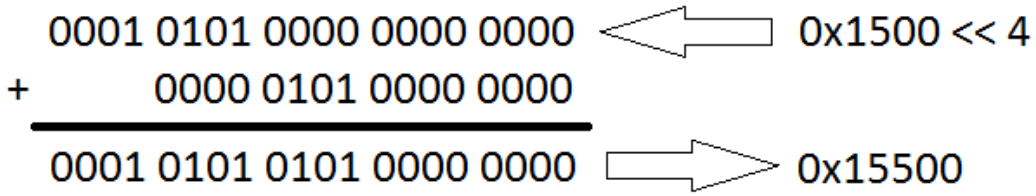
Δυστυχώς δεν υπάρχει σίγουρος τρόπος για την ενεργοποίηση της γραμμής αυτής. Διαφορετικοί κατασκευαστές έχουν διαφορετικές διαδικασίες για την ενεργοποίησή της. Υπάρχουν τουλάχιστον 4 βασικοί τρόποι για να πετύχουμε αυτό. Αυτό που κάνουμε λοιπόν είναι να δοκιμάσουμε να ενεργοποιήσουμε τη γραμμή με την πρώτη μέθοδο και μετά να ελέγξουμε αν η γραμμή έχει ενεργοποιηθεί με τη διαδικασία που περιγράψαμε προηγουμένως. Αν όχ,ι συνεχίζουμε με την δεύτερη μέθοδο κ.ο.κ μέχρι κάποια μέθοδος να πετύχει. Αν καμία μέθοδος δεν καταφέρει να ενεργοποιήσει την γραμμή τότε απλά ο φορτωτήςεκκίνησης εμφανίζει το αντίστοιχο μήνυμα στην οθόνη και η ροή εκτέλεσης του προγράμματος σταματάει.

Στην εργασία αυτή δοκιμάζουμε μόνο έναν από τους πιθανούς τρόπους που είναι και ο πιο διαδεδομένος. Μια σημαντική μελλοντική προσθήκη θα ήταν να δοκιμάζουμε με όλους τους τρόπους που περιγράφηκαν νωρίτερα. Κάτι ακόμα σημαντικό είναι ότι σύμφωνα με το[15] η σειρά με την οποία δοκιμάζουμε να ενεργοποιήσουμε την γραμμή πρέπει να είναι συγκεκριμένη.

Έστω λοιπόν ότι έχουμε καταφέρει με επιτυχία να ενεργοποιήσουμε την γραμμή. Τώρα πλέον έχουμε πρόσβαση σε όλες τις διαθέσιμες διευθύνσεις που μπορεί τοποθετήσει στον δίαυλο διευθύνσεων ο επεξεργαστής μόλις μεταβούμε σε κατάσταση 32-bitprotectedmode. Για την ώρα σε κατάσταση 16-bitrealmode έχουμε πρόσβαση μόνο στο πρώτο MB.

Για καλύτερη κατανόηση του φορτωτή εκκίνησης θα πρέπει να εξηγήσουμε συνοπτικά πως γίνεται η διευθυνσιοδότηση, όταν ο επεξεργαστής βρίσκεται σε κατάσταση 16-bitrealmode.Ο επεξεργαστής σε αυτήν την κατάσταση χρησιμοποιεί τους 16-bit καταχωρητές τμήματος που διαθέτει, μετατοπίζει αριστερά κατά 4 bit την τιμή που περιέχουν και προσθέτει στο αποτέλεσμα την αναφερόμενη διεύθυνση. Έτσι παράγεται η τελική διεύθυνση.

Στο παρακάτω σχήμα μπορούμε να δούμε ένα παράδειγμα. Η τιμή του καταχωρητή τμήματος είναι 0x1500 και η διεύθυνση τμήματος 0x500.Η φυσική διεύθυνση που παράγεται ως έξοδος είναι η 0x15500. Για περισσότερες λεπτομέρειες πάνω στην διευθυνσιοδότηση των 16-bit επεξεργαστών της οικογένειας x86 μπορείτε να δείτε εδώ[16].



Εικόνα 1. Διευθυνσιοδότηση μνήμης 16-bitrealmode.

Το επόμενο βήμα είναι να μεταφέρουμε τον ίδιο τον φορτωτή εκκίνησης σε κάποιο άλλο σημείο στην μνήμη. Ο λόγος που θέλουμε να το κάνουμε αυτό είναι επειδή δεν υπάρχει αρκετόςδιαθέσιμος χώρος για να φορτώσουμε τον πυρήνα. Όχι μόνο δεν έχουμε πρόσβαση σε διευθύνσεις μεγαλύτερες του 1MB αλλά επίσης μέρος αυτού του χώρου χρησιμοποιείται από το BIOS. Ακόμα χειρότερα, αυτός ο διαθέσιμος χώρος είναι κατακερματισμένος από τον ίδιο τον φορτωτή εκκίνησης ο οποίος βρίσκεται στη θέση 0x7C00.

Start	End	Size	description	Type	
Real mode address space (the first MiB)					
0x00000000	0x000003FF	1 KiB	Real Mode IVT (Interrupt Vector Table)	unusable in real mode	640 KiB RAM ("Low memory")
0x00000400	0x000004FF	256 bytes	BDA (BIOS data area)		
0x00000500	0x00007BFF	almost 30 KiB	Conventional memory	usable memory	
0x00007C00	0x00007DFF	512 bytes	Your OS BootSector		
0x00007E00	0x0007FFFF	480.5 KiB	Conventional memory		
0x00080000	0x0009FFFF	128 KiB	EBDA (Extended BIOS Data Area)	partially used by the EBDA	

0x000A0000	0x000BFFFF	128 KiB	Video display memory	hardware mapped	384 KiB System / Reserved ("Upper Memory")
0x000C0000	0x000C7FFF	32 KiB (typically)	Video BIOS	ROM and hardware mapped / Shadow RAM	
0x000C8000	0x000EFFFF	160 KiB (typically)	BIOS Expansions		
0x000F0000	0x000FFFFF	64 KiB	Motherboard BIOS		

Πίνακας 1. Οι περιοχές μνήμης που δεσμεύει το BIOS.

Πηγή:https://wiki.osdev.org/Memory_Map_(x86)

Μεταφέρουμε λοιπόν τον φορτωτή εκκίνησης στη θέση 0x100000. Μετά και από αυτή τη μεταφορά έχουμε στη διάθεση μας περίπου 510 KB ελεύθερου χώρου, ξεκινώντας από την διεύθυνση 0x500 όπως φαίνεται και στον πίνακα. Ο χώρος αυτός είναι αρκετός για να χωρέσει τον πυρήνα. Επειδή στην περίπτωση που το αρχείο του πυρήνα μεγαλώσει αρκετά και ξεπεράσει αυτό το όριο, δεν θα έχουμε καμία ειδοποίηση, έχουμε ορίσει μία παράμετρο στον συνδέτη ώστε να παράγει σφάλμα κατά τη φάση της παραγωγής του αρχείου. Αυτή η παράμετρος υπάρχει στο αρχείο *ldscript* και ορίζει ότι το τμήμα του κώδικα δεν μπορεί να ξεπερνάει τα 510 KB. Μετά από τη μεταφορά του φορτωτή εκκίνησης στην νέα θέση η εκτέλεση του προγράμματος συνεχίζεται από εκεί με μία εντολή άλματος.

Το επόμενο βήμα είναι η μεταφορά του πυρήνα από το αποθηκευτικό μέσο στη μνήμη, στη θέση 0x500. Αυτό γίνεται με της κλήσεις που παρέχει το BIOS. Συγκεκριμένα, με την κλήση INT 0x13. Με την ολοκλήρωση της μεταφοράς ελέγχουμε εάν η μεταφορά ολοκληρώθηκε με επιτυχία. Αν δεν ολοκληρώθηκε με επιτυχία εμφανίζεται αντίστοιχο μήνυμα στην οθόνη και η εκτέλεση του προγράμματος σταματάει.

Μετά τη φόρτωση του πυρήνα ακολουθούν διαδοχικές κλήσεις στο BIOS που επιστρέφουν τις ελεύθερες και δεσμευμένες περιοχές της φυσικής μνήμης (INT 0x15). Έτσι σχηματίζουμε έναν στατικό πίνακα όπου κάθε εγγραφή του περιγράφει και μία περιοχή μνήμης. Φυσικά κάθε περιοχή μπορεί να έχει διαφορετικό μέγεθος. Ο πίνακας αυτός είναι ιδιαίτερα σημαντικός για τον διαχειριστή μνήμης όταν θα εκτελεστεί από τον πυρήνα αργότερα. Φυσικά ο πυρήνας πρέπει να γνωρίζει τη διεύθυνση του πίνακα αυτού που είναι η 0x100204. Από το άθροισμα του μεγέθους όλων αυτών των

περιοχών μπορούμε να υπολογίσουμε και την ποσότητα της εγκατεστημένης μνήμης. Αυτό γίνεται από το υποσύστημα διαχείρισης μνήμης στον πυρήνα αργότερα.

Το τελευταίο βήμα για να μεταβούμε στον πυρήνα και να ξεκινήσει το λειτουργικό είναι να αλλάζουμε την κατάσταση του επεξεργαστή από 16-bitrealmodeσε 32-bitprotectedmode. Για αυτό τον σκοπό υπάρχει κατάλληλη δομή δεμένων ενσωματωμένη στον φορτωτή εκκίνησης. Για περισσότερες πληροφορίες σχετικά με αυτή τη δομή μπορείτε να δείτε εδώ[17]. Αυτή η δομή φορτώνεται στη θέση μνήμης 0x00 και στη συνέχεια με την εντολή LGDT ενημερώνουμε τον επεξεργαστή για την θέση αυτής της δομής στη μνήμη. Τέλος, θέτουμε το 1^{ο}bit του καταχωρητή CR0 ίσο με ένα και με μία εντολή άλματος ξεκινάμε την εκτέλεση του πυρήνα.

1.3 Ο πυρήνας και τα υποσυστήματα του

Το λειτουργικό σύστημα ως λογισμικό πρέπει να διαχειρίζεται το υλικό και να παρέχει μία ομαλή διεπαφή στα προγράμματα χρήστη. Το πλήθος των διαφορετικών συσκευών που διαχειρίζεται ταυτόχρονα, οι αφαιρέσεις που πρέπει να δημιουργεί για το λογισμικό υψηλότερου επιπέδου (όπως το σύστημα αρχείων) και οι ανάγκες σε μνήμη που μπορεί να έχει οποιαδήποτε χρονική στιγμή κάποια διεργασία, είναι μόνο ένα μικρό μέρος από τις υπηρεσίες που πρέπει να παρέχει ένα λειτουργικό σύστημα. Το συμπέρασμα που προκύπτει από την παραπάνω παρατήρηση είναι ότι ο μεγάλος αριθμός διαφορετικών λειτουργιών που πρέπει να υποστηρίζει το λειτουργικό σύστημα αυξάνουν την πολυπλοκότητα του συστήματος, ειδικά όταν όλες αυτές οι λειτουργίες αλληλεπιδρούν μεταξύ τους. Μία μέθοδος για να ελαχιστοποιηθεί η πολυπλοκότητα του συστήματος είναι να χωρίσουμε το λειτουργικό σε μικρότερα υποσυστήματα όπου το κάθε ένα από αυτά θα διεκπεραιώνει μια συγκεκριμένη λειτουργία του συνολικού συστήματος. Σε αυτήν την ενότητα παρουσιάζονται όλα τα υποσυστήματα και οι λειτουργίες τους.

1.3.1 Ο χρονοπρογραμματιστής και οι διεργασίες

Στην παρούσα υλοποίηση οι διεργασίες δεν είναι τίποτα περισσότερο από μία δομή (struct) στη C.Η δομή αυτή περιέχει μία συλλογή δεδομένων για τη διεργασία που αναπαριστά. Για κάθε νέα διεργασία που δημιουργείται λοιπόν δεσμεύεται χώρος δυναμικά στη μνήμη. Αντίστοιχα με τον τερματισμό μίας διεργασίας ο χώρος αυτός απελευθερώνεται και πάλι.

Κάθε μία από αυτές τις διεργασίες μπορεί να βρίσκεται μέσα σε μία από τρείς συνδεδεμένες λίστες που διατηρεί συνεχώς το λειτουργικό σύστημα, ανάλογα με την κατάσταση στην οποία βρίσκεται. Υπάρχει μία συνδεδεμένη λίστα με διεργασίες που είναι διαθέσιμες προς εκτέλεση, μία λίστα με διεργασίες που περιμένουν τον τερματισμό άλλων διεργασιών για να εκτελεστούν και μία λίστα με διεργασίες που έχουν μπλοκαριστεί από το υλικό του υπολογιστή.

Ανά μερικά κλάσματα του δευτερολέπτου ένας μετρητής του επεξεργαστή (Programmable Interval Timer ή PIT) προκαλεί διακοπές. Τότε ο αντίστοιχος χειριστής διακοπών εκτελεί τον χρονοπρογραμματιστή. Ο χρονοπρογραμματιστής με τη σειρά του αποθηκεύει την κατάσταση της τρέχουσας διεργασίας, ουσιαστικά όλους τους καταχωρητές συμπεριλαμβανομένου φυσικά και του καταχωρητή IP που περιέχει τη διεύθυνση της επόμενης προς εκτέλεση εντολής πριν τη διακοπή του μετρητή. Όλες αυτές οι πληροφορίες αποθηκεύονται στη δομή που αναπαριστά τη διεργασία στη μνήμη. Έπειτα ο χρονοπρογραμματιστής ελέγχει τη λίστα με τις διαθέσιμες προς εκτέλεση διεργασίες και εκτελεί την επόμενη, ενημερώνει δηλαδή όλους τους καταχωρητές με τις τιμές που είχε αποθηκεύσει νωρίτερα. Τέλος, ή νέα διεργασία εκτελείται ωσότου ο μετρητής του επεξεργαστή να προκαλέσει νέα διακοπή και η επόμενη διεργασία να ξεκινήσει την εκτέλεση της.

1.3.2 Ο διαχειριστής μνήμης

Οι διεργασίες στον χώρο του χρήστη άλλα και ο ίδιος ο πυρήνας κατά τον χρόνο εκτέλεσης τους είναι βέβαιο ότι κάποια στιγμή θα χρειαστεί να δεσμεύσουν μνήμη από το σύστημα για να αποθηκεύσουν δεδομένα. Είναι πολύ σημαντικό το λειτουργικό να γνωρίζει ανά πάσα στιγμή την ελεύθερη και τη δεσμευμένη ποσότητα μνήμης στον υπολογιστή. Πρέπει επομένως να παρέχει μία διαδικασία η οποία θα δεσμεύει μνήμη καθώς και μία που θα αποδεσμεύει τη μνήμη για να μπορέσει να ξανά χρησιμοποιηθεί αργότερα.

Άλλες σημαντικές υπηρεσίες που πρέπει να παρέχονται από το λειτουργικό σύστημα είναι η προστασία της μνήμης από άλλες διεργασίες (να έχουν δηλαδή άλλες διεργασίες πρόσβαση στα δεδομένα άλλων διεργασιών ούτε για ανάγνωση/εγγραφή ούτε για εκτέλεση), ο ενιαίος χώρος διευθύνσεων (ανεξάρτητα από το αν υπάρχουν άλλες διεργασίες στη μνήμη) και ο διαχωρισμός των δεδομένων από της εκτελέσιμες εντολές. Με λίγα λόγια αφού φορτωθούν οι εντολές ενός προγράμματος στη μνήμη να μην μπορεί να τροποποιηθεί το περιεχόμενο τους από οποιαδήποτε διεργασία μέχρι το πρόγραμμα αυτό να τερματιστεί. Επίσης αντίστροφα να μην μπορεί να γίνει εκτέλεση τον δεδομένων ενός προγράμματος από τον επεξεργαστή σαν να ήταν εντολές προγράμματος. Για περισσότερες πληροφορίες σχετικά με τα προβλήματα που πρέπει να λάβουμε υπ’ όψιν μας όταν σχεδιάζουμε το υποσύστημα μνήμης αλλά και με ποιους πιθανούς τρόπους μπορούν να λυθούν μπορείτε να δείτε εδώ[18]. Ένας μηχανισμός που προσφέρει λύση σε όλα τα παραπάνω προβλήματα είναι η σελιδοποίηση (paging) και η εικονική μνήμη (virtualmemory).

Το παρόν λειτουργικό δυστυχώς δεν υποστηρίζει σελιδοποίηση και εικονική μνήμη αν και η οικογένεια επεξεργαστών x86 τα υποστηρίζει[19]. Ωστόσο το υποσύστημα μνήμης έχει σχεδιαστεί με τέτοιο τρόπο ώστε αν μελλοντικά θελήσουμε να υποστηρίξουμε τη λειτουργία αυτή να μπορεί να γίνει με ελάχιστες αλλαγές.

Όπως είχαμε δει και νωρίτερα κατά τη εκκίνηση αποθηκεύσαμε σε γνωστή θέση έναν στατικό πίνακα που περιέχει όλες τις περιοχές μνήμης που είναι δεσμευμένες και όλες τις ελεύθερες περιοχές. Κατά την αρχικοποίηση του υποσυστήματος μνήμης σχεδιάζουμε ένα χάρτη bit με βάση αυτόν τον στατικό πίνακα.Κάθε bitαναπαριστά μία περιοχή μνήμης από 4096 byte που ονομάζουμε πλαίσιο[20]. Όταν το bit έχει τιμή 1 τότε η αντίστοιχη περιοχή θεωρείται δεσμευμένη ενώ όταν έχει τιμή 0 θεωρείται ελεύθερη. Αφού η μέγιστη ποσότητα μνήμης που μπορεί να υποστηριχτεί σε 32 protectedmodeείναι 4GB= 2^{32}, αφού χωρίσαμε τη μνήμη σε περιοχές των 4096 byte= 2^{12}και αφού με κάθε bytεμπορούμε να αναπαραστήσουμε 8 = 2^3 περιοχές θα χρειαστούμε $2^{32}/2^{12}/ 2^3= 2^{17}$bytesή 128 KB.

Τέλος κατασκευάζουμε δύο κλήσεις συστήματος για δέσμευση-αποδέσμευση μνήμης. Η πρώτη κλήση που δεσμεύει μνήμη ως παράμετρο δέχεται μία διεύθυνση βάσης και το πλήθος τον πλαισίων μνήμης που θέλουμε να δεσμεύσουμε. Ελέγχει τον χάρτη bitαν η συγκεκριμένη περιοχή μνήμης είναι ελεύθερη ενημερώνει τον χάρτη bitότι αυτή η περιοχή μνήμης δεν είναι ελεύθερη πλέον και επιστρέφει έναν δείκτη προς αυτή την περιοχή. Αν η περιοχή αυτή δεν είναι ελεύθερη δεν γίνεται καμία αλλαγή στον χάρτη bit και επιστρέφει NULL. Αντίστροφα η δεύτερη κλήση συστήματος αποδεσμεύει μία περιοχή μνήμης ενημερώνοντας τον χάρτη bitκατάλληλα.

Έτσι με αυτόν τον τρόπο το λειτουργικό μπορεί να εκχωρεί δυναμικά μπλοκ μνήμης σε άλλα υποσυστήματα και διεργασίες χωρίς τον κίνδυνο αλληλοεπικάλυψης. Με λίγα λόγια το λειτουργικό διαχειρίζεται τη μνήμη σε επίπεδο πλαισίων μνήμης των 4096 byte. Επειδή όμως μπορεί διάφορες διεργασίες ή και ο ίδιος ο πυρήνας να χρειάζονται πολύ λιγότερο χώρο σε κάποιες περιπτώσεις για αποθήκευση δεδομένων, για να αποφευχθεί η σπατάλη, παρέχονται συναρτήσεις που οι ίδιες εσωτερικά δεσμεύουν πλαίσια μνήμης μέσω των κλήσεων συστήματος και διαχειρίζονται αυτόν τον χώρο με διαφορετικό τρόπο δίνοντας τη δυνατότητα για δέσμευση μνήμης σε επίπεδο byte. Μία τέτοια συνάρτηση είναι η *malloc()*.

1.3.3 Ο χειριστής διακοπών

Το υλικό, ο ίδιος ο επεξεργαστής, η ακόμα και κάποια διεργασία μπορεί να προκαλέσουν ανά πάσα στιγμή μία διακοπή [21]. Κάθε διακοπή πρέπει να έχει έναν αντίστοιχο χειριστή δηλαδή πρακτικά ένα σύνολο εντολών που πρόκειται να εκτελεστεί εκείνη τη χρονική στιγμή για τη διαχείριση της. Έτσι λοιπόν ο κάθε χειριστής διακοπής αποτελεί μέρος του αντίστοιχου υποσυστήματος ή οδηγού συσκευών που την υλοποιεί. Ωστόσο το λειτουργικό σύστημα παρέχει μία συνάρτηση η οποία δέχεται ως παράμετρο μία διεύθυνση προς έναν τέτοιο χειριστή και τον αριθμό διακοπής που εξυπηρετεί και ενημερώνει την αντίστοιχη δομή δεδομένων. Ο επεξεργαστής διατηρεί έναν δείκτη σε έναν καταχωρητή που δείχνει στην αρχή αυτής της δομής.Η δομή αυτή είναι ένα διάνυσμα από άλλους δείκτες που δείχνουν στην αρχή του κάθε χειριστή διακοπής. Ο αριθμός διακοπής φανερώνει την θέση της αντίστοιχης εγγραφής μέσα στο διάνυσμα.

Κατά την εκκίνηση του πυρήνα γίνονται πολλές κλήσης προς αυτήν την συνάρτηση ώστε να αρχικοποιηθούν όλες οι διακοπές που προκαλεί ο επεξεργαστής. Έπειτα, κατά τη φάση της φόρτωσης οδηγών γίνονται επίσης κλήσεις από τους οδηγούς συσκευών.

Οι χειριστές διακοπών που προκαλούνται από τον επεξεργαστή βρίσκονται στο αρχείο *exceptions.c* και σε αυτήν την έκδοση απλά εμφανίζουν μήνυμα και σταματούν τη λειτουργία του πυρήνα. Σε μελλοντική αναβάθμιση θα μπορούσαν να ελέγχουν αν η διακοπή προκλήθηκε από κάποια διεργασία (π.χ. διαίρεση με το 0) και να στέλνουν ένα σήμα σε αυτήν την διεργασία. Σε περίπτωση που η διεργασία δεν μπορεί να διαχειριστεί αυτήν την εξαίρεση, την τερματίζει ο πυρήνας.

1.3.4 Ο μηχανισμός παγίδευσης

Αφού κατά την εκκίνηση το λειτουργικό σύστημα κάνει τις απαραίτητες αρχικοποιήσεις θα μεταβιβάσει τον έλεγχο στην πρώτη διεργασία αλλάζοντας όμως πρώτα την κατάσταση του επεξεργαστή από κατάσταση πυρήνα σε κατάσταση χρήστη. Συγκεκριμένα, οι επεξεργαστές της οικογένειαςx86 έχουν 4 επίπεδα λειτουργίας (protectionlevel0, 1, 2, 3) αλλά από αυτά τα τέσσερα εμείς χρησιμοποιούμε μόνο τα επίπεδα 1 και το 3. Για περισσότερες πληροφορίες σχετικά με τα επίπεδα προστασίας στους επεξεργαστές της οικογένειαςx86 μπορείτε να δείτε εδώ [22]Το επίπεδο 0 είναι αυτό με τα περισσότερα δικαιώματα και σε αυτό εκτελείται ο πυρήνας. Από την άλλη το επίπεδο 3 είναι αυτό με τα λιγότερα δικαιώματα και σε αυτό εκτελούνται οι διεργασίες χρήστη. Έτσι όπως τα λειτουργικά Windows και Linux έχουν μόνο δύο επίπεδα προστασίας, με παρόμοια φιλοσοφία και η δική μας υλοποίηση έχει μόνο δύο επίπεδα. Δύο σημαντικά προβλήματα με τη χρήση περισσότερων επιπέδων θα ήταν ότι: πρώτον η υλοποίηση περισσότερων επιπέδων θα αύξανε την πολυπλοκότητα της υλοποίησης και δεύτερον σε περίπτωση που θα θέλαμε να μεταφέρουμε το λειτουργικό σε κάποιον άλλο επεξεργαστή θα χρειαζόταν να κάνουμε στον κώδικα σημαντικά περισσότερες αλλαγές καθώς οι περισσότεροι επεξεργαστές υποστηρίζουν δύο επίπεδα προστασίας[23]. Για παράδειγμα τα Linux υποστηρίζουν πολλούς επεξεργαστές εκτός της οικογένειαςx86. Για τον ίδιο λόγο σύμφωνα με το [24]και τα Windows υλοποιήθηκαν με αυτόν τον τρόπο έτσι ώστε στο μέλλον να μπορούν να τρέξουν σε διαφορετικούς επεξεργαστές αν ποτέ χρειαστεί.

Οι οικογένεια επεξεργαστών x86 υποστηρίζει διάφορους τρόπους για μετάβαση σε κατάσταση πυρήνα[25]. Στην εργασία αυτή η μετάβαση γίνεται με μία διακοπή λογισμικού. Συγκεκριμένα με τον αριθμό 255 (int 255).Ο χειριστής της διακοπής ελέγχει την τιμή στον καταχωρητή eax και δίνει τον έλεγχο στην αντίστοιχη κλήση συστήματος. Οι παράμετροι που θέλουμε να περάσουμε στην κλήση αποθηκεύονται στους υπόλοιπους καταχωρητές. Στον τέλος της κλήσης συστήματος με την εντολή iret ξανά γίνεται μετάβαση σε κατάσταση χρήστη.

1.3.5 Το σύστημα αρχείων

Σημαντικό μέρος στο λειτουργικό έχει και το σύστημα αρχείων.Στην υλοποίηση της παρούσας εργασίας παρέχεται η δυνατότητα για σύνδεση με οποιοδήποτε σύστημα αρχείων. Θα πρέπει βέβαια να γραφτούν οι αντίστοιχες, συμβατές με το λειτουργικό, βιβλιοθήκες. Μάλιστα εφόσον υπάρχει ή αντίστοιχη βιβλιοθήκη η σύνδεση μπορεί να γίνει κατά τον χρόνο εκτέλεσης.

Το βασικό σύστημα αρχείων που έχει υλοποιηθεί για την εργασία είναι το FAT32. Υποστηρίζονται οι ενέργειες για ανάγνωση, εγγραφή, και δημιουργία αρχείων. Υποστηρίζονται ακόμα λειτουργίες για επιστροφή όλων των ονομάτων των αρχείων ενός καταλόγου καθώς και λειτουργία που ελέγχει εάν ένας κατάλογος ή ένα αρχείο υπάρχει ή όχι. Δυστυχώς δεν υποστηρίζονται ενέργειες για διαγραφή/δημιουργία καταλόγου.

Για ταχύτερη ανάγνωση και εγγραφή αλλά και για ταχύτερη αναζήτηση αρχείων χρησιμοποιούνται κρυφές μνήμες.Πρώτα γίνεται έλεγχος στην κρυφή μνήμη (η κρυφή μνήμη σε αυτήν την περίπτωση είναι ένας bufferστην RAM) και αν δεν βρεθεί το μπλοκ που αναζητούμε εκεί, τότε γίνεται ανάγνωση από τον πιο αργό δίσκο. Στη συνέχεια, τα δεδομένα που διαβάστηκαν από τον δίσκο αποθηκεύονται στην κρυφή μνήμη για μελλοντική χρήση. Αν στην ίδια θέση της κρυφής μνήμης υπήρχαν άλλα δεδομένα ελέγχουμε αν αυτά είχαν τροποποιηθεί ή απλώς αναγνώστηκαν. Σε περίπτωση που είχαν τροποποιηθεί, θα πρέπει να ξαναεγγραφούν στο αποθηκευτικό μέσο πριν μεταφέρουμε το νέο μπλοκ μνήμης.

2. Η διεπαφή χρήστη του λειτουργικού

2.1 Προγραμματιστική διεπαφή

Ένα λειτουργικό σύστημα χωρίς προγραμματιστική διεπαφή θα ήταν πρακτικά άχρηστο. Δεν θα μπορούσε να παρέχει ούτε τις πιο βασικές λειτουργίες στα προγράμματα χρήστη, όπως για παράδειγμα τη δυνατότητα για ανάγνωση και εγγραφή αρχείων ή βασικές λειτουργίες εισόδου/εξόδου από πληκτρολόγιο και οθόνη.

Στα Windows η αντίστοιχη προγραμματιστική διεπαφή ονομάζεται Windows API [26] ενώ στα Linux πιο άμεσα υποστηρίζεται ένα σύνολο συναρτήσεων (systemcallwrappers) οι οποίες με την σειρά τους καλούν τις αντίστοιχες κλήσεις συστήματος. Στην παρούσα εργασία ακολουθούμε την ίδια φιλοσοφία.Ταυτόχρονα μέρος της πρότυπης βιβλιοθήκης της C έχει υλοποιηθεί και επομένως υπάρχει η δυνατότητα να γραφούν προγράμματα που θα εκτελούνται πάνω στο συγκεκριμένο λειτουργικό χωρίς ο προγραμματιστής να γνωρίζει της κλήσεις συστήματος. Έτσι για παράδειγμα ο προγραμματιστής μπορεί να γράψει ένα πρόγραμμα με τις βασικές συναρτήσεις εισόδου και εξόδου της C η οποίες με τη σειρά τους καλούν εσωτερικά τις κατάλληλες κλήσεις συστήματος όπου και όταν χρειάζεται.

Τέλος υπάρχει διεπαφή και για δημιουργία οδηγών συσκευών. Και εδώ σε κάποιο βαθμό η φιλοσοφία είναι παρόμοια με τα Linux.Έτσι, κατά τον χρόνο εκτέλεσης του λειτουργικού αρχεία οδηγών συσκευών μπορούν δυναμικά να φορτωθούν από το σύστημα αρχείων και να ενσωματωθούν στον πυρήνα. Στις ακόλουθες υποενότητες γίνεται μία περιγραφή των παραπάνω προγραμματιστικών διεπαφών.

2.1.1 Κλήσεις συστήματος

Καθώς όλα τα προγράμματα πέραν του ίδιου του λειτουργικού εκτελούνται σε κατάσταση χρήστη για λόγους ασφάλειας και σταθερότητας του συστήματος, έχουν περιορισμένες δυνατότητες. Για να αλληλεπιδρούν με τη μνήμη, το υλικό και τις άλλες διεργασίες, ένα σύνολο από κλήσεις συστήματος έχουν υλοποιηθεί. Έτσι λοιπόν περνώντας τις κατάλληλες παραμέτρους στους καταχωρητές και καλώντας στη συνέχεια μία εντολή παγίδευσης, ο επεξεργαστής εισέρχεται σε κατάσταση πυρήνα με ελεγχόμενο τρόπο, εκτελεί την αντίστοιχη κλήση συστήματος και τέλος επιστρέφει σε κατάσταση χρήστη δίνοντας τον έλεγχο στο πρόγραμμα που την κάλεσε. Φυσικά ο προγραμματιστής των εφαρμογών στον χώρο του χρήστη δεν χρειάζεται να γνωρίζει αυτές τις λεπτομέρειες. Ένα σύνολο από συναρτήσεις παρέχεται ως μέρος της υλοποίησης με συναρτήσεις (functionwrappers) που αποκρύπτουν αυτές τις λεπτομέρειες προσφέροντας στον προγραμματιστή μία απλούστερη διεπαφή. Έτσι ο προγραμματιστής δεν χρειάζεται να γνωρίζει για παράδειγμα ποιους καταχωρητές πρέπει να ενημερώσει πριν εκτελέσει μία κλήση συστήματος. Το μόνο που πρέπει να γνωρίζει είναι οι συναρτήσεις που καλούν αυτές τις κλήσεις και υπάρχουν στο αρχείο

*sysCallLib.h.*Έτσι εμμέσως τα προγράμματα στον χώρο του χρήστη έχουν την δυνατότητα να καλέσουν ένα πλήθος από κλήσεις συστήματος για είσοδο/έξοδο, δέσμευση μνήμης, δημιουργία νέων διεργασιών κ.α.Στον ακόλουθο πίνακα εμφανίζονται όλες οι συναρτήσεις τον κλήσεων συστήματος μαζί με μία συνοπτική περιγραφή για την λειτουργία τους.

Πίνακας 2. Οι κλήσεις συστήματος του λειτουργικού

Ονομασία κλήσης	Περιγραφή
ScPrintChar	Τυπώνει έναν χαρακτήρα στην οθόνη
freePhysicalPages	Αποδεσμεύει πλαίσια μνήμης
memAlloc	Δεσμεύει μνήμη
createProcces	Ξεκινάει μία διεργασία
exitProc	Τερματίζει την διεργασία που την κάλεσε
waitProc	Μπλοκάρει τη διεργασία από τον χρονοπρογραμματιστή
readFile	Ανάγνωση δεδομένων από αρχείο
writeFile	Εγγραφή δεδομένων σε αρχείο
openFile	Άνοιγμα αρχείου
seekFile	Μετακίνηση του κέρσορα του αρχείου
closeFile	Κλείσιμο αρχείου
getMessage	Ελέγχει για το επόμενο μήνυμα (event) στην ουρά
setProcProperties	Αλλάζει βασικά χαρακτηριστικά της διεργασίας(π.χ. Αν έχει διεπαφή χρήστη)
setActiveProc	Ορίζει ποια διεργασία βρίσκεται στο παρασκήνιο
Draw	Τυπώνει πολλούς χαρακτήρες στην οθόνη. Σχεδιάστηκε έτσι ώστε να δουλεύει και με γραφικά με κάποια μικρή τροποποίηση
fileExists	Τυπώνει πολλούς χαρακτήρες στην οθόνη. Σχεδιάστηκε έτσι ώστε να δουλεύει και με γραφικά με κάποια μικρή τροποποίηση
getFileAttr	Επιστρέφει τα μεταδεδομένα ενός αρχείου. Για την ώρα μόνο το μέγεθος του
createFile	Δημιουργεί ένα νέο αρχείο
loadDriver	Φορτώνει και εκκινεί έναν οδηγό συσκευών

2.1.2 Πρότυπη βιβλιοθήκη της C

Εκτός από τις κλήσεις συστήματος και σημαντικό μέρος της πρότυπης βιβλιοθήκης της C(C standard library) έχει υλοποιηθεί με τις παραπάνω διαδικασίες. Έχει υλοποιηθεί ως στατική βιβλιοθήκη στον χώρο του χρήστη. Έτσι κατά την μεταγλώττιση των προγραμμάτων για το παρόν λειτουργικό ο προγραμματιστής μπορεί να χρησιμοποιήσει τη βιβλιοθήκη αυτή για να δημιουργήσει προγράμματα χωρίς να έχει μελετήσει τις κλήσεις συστήματος του λειτουργικού. Ταυτόχρονα αυτά τα προγράμματα θα είναι σε θέση να ξανά μεταγλωττιστούν και να εκτελεστούν και σε άλλες πλατφόρμες για τις οποίες υπάρχει υλοποίηση της πρότυπης βιβλιοθήκης (source-compatible). Αντίστροφα υπάρχει και η δυνατότητα προγράμματα που έχουν ήδη γραφτεί για διαφορετικές πλατφόρμες και χρησιμοποιούν την πρότυπη βιβλιοθήκη να ξανά μεταγλωττιστούν για τη συγκεκριμένη υλοποίηση.

Για την υλοποίηση τους, οι βασικές συναρτήσεις της πρότυπης βιβλιοθήκης για είσοδο και έξοδο όπως η printf() και η gets() εσωτερικά χρησιμοποιούν τις κλήσεις συστήματος που παρέχει το λειτουργικό. Άλλες συναρτήσεις, όπως για παράδειγμα η strncpy() δεν εκτελούν κάποια κλήση συστήματος καθώς δεν υπάρχει αυτή η ανάγκη.

2.1.3 Οδηγοί συσκευών και επέκταση του πυρήνα

Αν και οι κλήσεις συστήματος και η πρότυπη βιβλιοθήκη προσφέρουν πολλές υπηρεσίες στα προγράμματα χρήστη, δεν δίνουν τη δυνατότητα για συγγραφή οδηγών συσκευών. Καθώς αυτό το λειτουργικό σύστημα είναι συμβατό με τους υπολογιστές IBMPC για τους οποίους υπάρχουν εκατοντάδες περιφερειακές συσκευές πρέπει να παρέχει έναν τρόπο για διασύνδεση με αυτές. Το να έχει όλους τους οδηγούς συσκευών ενσωματωμένους κατά τον χρόνο μεταγλώττισης δεν θα ήταν πρακτικό για πολλούς λόγους. Έτσι λοιπόν, όπως σε όλα τα λειτουργικά συστήματα της ίδιας κατηγορίας, υπάρχει μηχανισμός για φόρτωση τον οδηγών συσκευών κατά τον χρόνο εκτέλεσης έτσι υπάρχει και εδώ. Για τον σκοπό αυτό όλες οι διευθύνσεις τον συναρτήσεων του λειτουργικού που είναι απαραίτητες για τον οδηγό υπάρχουν σε έναν πίνακα σε μία γνωστή θέση στην μνήμη του υπολογιστή.Μόλις ο οδηγός συσκευής φορτωθεί από το σύστημα αρχείων και εκτελεστεί μπορεί να συνδεθεί δυναμικά με αυτές τις συναρτήσεις και στη συνέχεια να τις εκτελέσει. Έτσι ο προγραμματιστής του οδηγού συσκευών μπορεί να γράφει το πρόγραμμα σαν να ήταν μέρος του πυρήνα καλώντας οποιαδήποτε συνάρτηση του πυρήνα. Οι οδηγοί συσκευών πρέπει να εκτελούνται στον χώρο του πυρήνα. Έτσι λοιπόν οι οδηγοί συσκευών ενσωματώνονται με τον πυρήνα και εκτελούνται σαν να είναι μέρος αυτού.Ένα παράδειγμα αυτού του μηχανισμού είναι ο οδηγός του πληκτρολογίου που είναι μέρος αυτής της εργασίας. Τα αντίστοιχα αρχεία υπάρχουν στον φάκελο *Os_keyboardDriver.*

Παρατηρούμε επομένως ότι μέσω αυτού του μηχανισμού όχι μόνο μπορούμε να δημιουργήσουμε οδηγούς συσκευών αλλά και να επεκτείνουμε τη λειτουργία του πυρήνα φορτώνοντας αρχεία εκτελέσιμου κώδικα εικονικών οδηγών που εκτελούνται στον χώρο του πυρήνα. Για παράδειγμα θα μπορούσαμε να έχουμε ένα τέτοιο αρχείο εκτελέσιμου κώδικα το οποίο περιέχει ένα νέο σύστημα αρχείων και να το φορτώνουμε κατά βούληση όταν υπάρχει η ανάγκη για ανάγνωση και εγγραφή από ένα αποθηκευτικό μέσο το οποίο διατηρεί τα αρχεία του σε αυτό το σύστημα αρχείων.Παρόμοιος μηχανισμός υπάρχει και στα Linuxμε τα kernelmodules[27].

2.2 Διεπαφή χρήστη

Πέρα από της δυνατότητες που παρέχονται από την προγραμματιστική διεπαφή απαραίτητη είναι και η υποστήριξη κάποιων βασικών λειτουργιών για τον χρήστη. Χωρίς την ύπαρξη κάποιον βασικών προγραμμάτων στον χώρο του χρήστη η υλοποίηση θα ήταν ημιτελής. Τα προγράμματα αυτά αν και δεν είναι μέρος του πυρήνα είναι απαραίτητα για την ύπαρξη μίας διεπαφής μεταξύ του χρήστη και του πυρήνα.

Το βασικότερο πρόγραμμα που εκτελείται στον χώρο του χρήστη και αποτελεί μέρος του λειτουργικού είναι το κέλυφος με το οποίο αλληλεπιδρά ο χρήστης μέσω εντολών. Εκτός του κελύφους κάποια επιπλέον προγράμματα στον χώρο του χρήστη έχουν υλοποιηθεί και θεωρούνται μέρος του λειτουργικού.Πρόκειται για μία υπηρεσία (ένα πρόγραμμα που τρέχει συνεχώς στο παρασκήνιο χωρίς διεπαφή χρήστη) και είναι υπεύθυνο να ορίζει ποια εφαρμογή είναι στο προσκήνιο. Ακόμα, μέρος του λειτουργικού θεωρείται και μία εφαρμογή που κατά την εκκίνηση του υπολογιστή ορίζει ποιοι οδηγοί συσκευών και ποιες εφαρμογές θα ξεκινήσουν αυτόματα.

Εκτός των εφαρμογών του λειτουργικού έχουν συμπεριληφθεί και κάποιες επιπλέον εφαρμογές που δεν αποτελούν μέρος του. Ο λόγος που αυτές οι εφαρμογές συμπεριλήφθησαν ήταν αρχικά για δοκιμαστικούς σκοπούς (ώστε να ήμαστε βέβαιοι ότι ο πυρήνας που εξυπηρετεί αυτές τις εφαρμογές είναι λειτουργικός) και στην συνέχειαδιατηρήθηκανώστε μπορεί να γίνει μία πιο ολοκληρωμένη παρουσίαση της εργασίας.

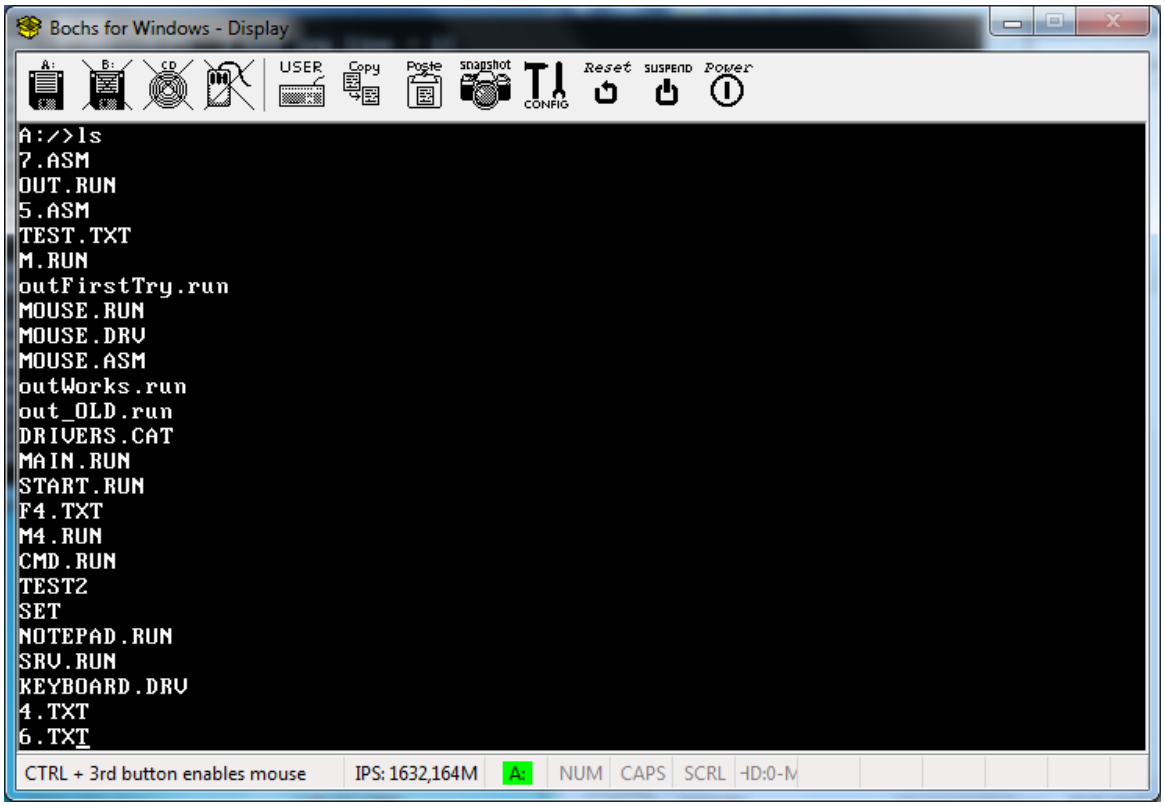
2.2.1 Γραμμή εντολών

Η γραμμή εντολών ανοίγει αυτόματα κατά την εκκίνηση. Μέσω της γραμμής εντολών ο χρήστης μπορεί να εκτελέσει διάφορες ενέργειες. Οι παρακάτω εντολές υποστηρίζονται:

- **echo:** Δέχεται ως όρισμα ένα αλφαριθμητικό και το εμφανίζει στην οθόνη
- **cls:**Καθαρίζει την οθόνη από προηγούμενη είσοδο. Δεν δέχεται κάποιο όρισμα
- **cd:**Αλλάζει τον τρέχοντα φάκελο. Δέχεται ως όρισμα τη διαδρομή (path) του νέου φακέλου
- **ls:**Εμφανίζει όλα τα αρχεία του φακέλου που δέχεται ως όρισμα
- **drvload:**Φορτώνει και εκκινεί έναν οδηγό συσκευών. Δέχεται ως όρισμα το όνομα του αρχείου του οδηγού
- **fcreate:**Δημιουργεί ένα νέο αρχείο. Δέχεται ως όρισμα το όνομα του αρχείου

Εκτός από τις παραπάνω εντολές υπάρχει φυσικά και η δυνατότητα να εκκινήσουμε οποιαδήποτε εφαρμογή γράφοντας το όνομά ακολουθούμενο από τα ορίσματα που δέχεται το πρόγραμμα και θέλουμε να περάσουμε στην εφαρμογή. Ένας δείκτης προς τα ορίσματα αυτά περνιέται στην στοίβα της εφαρμογής όταν αυτή φορτώνεται στη μνήμη. Έτσι, για παράδειγμα, σε ένα πρόγραμμα γραμμένο σε C στον

ορισμό της κύριας συνάρτησης *void _main(char params[])* στα ορίσματα θα είχαμε πρόσβαση μέσω της μεταβλητής *params.*



Εικόνα 2. Η γραμμή εντολών

2.2.2 Προγράμματα χρήστη

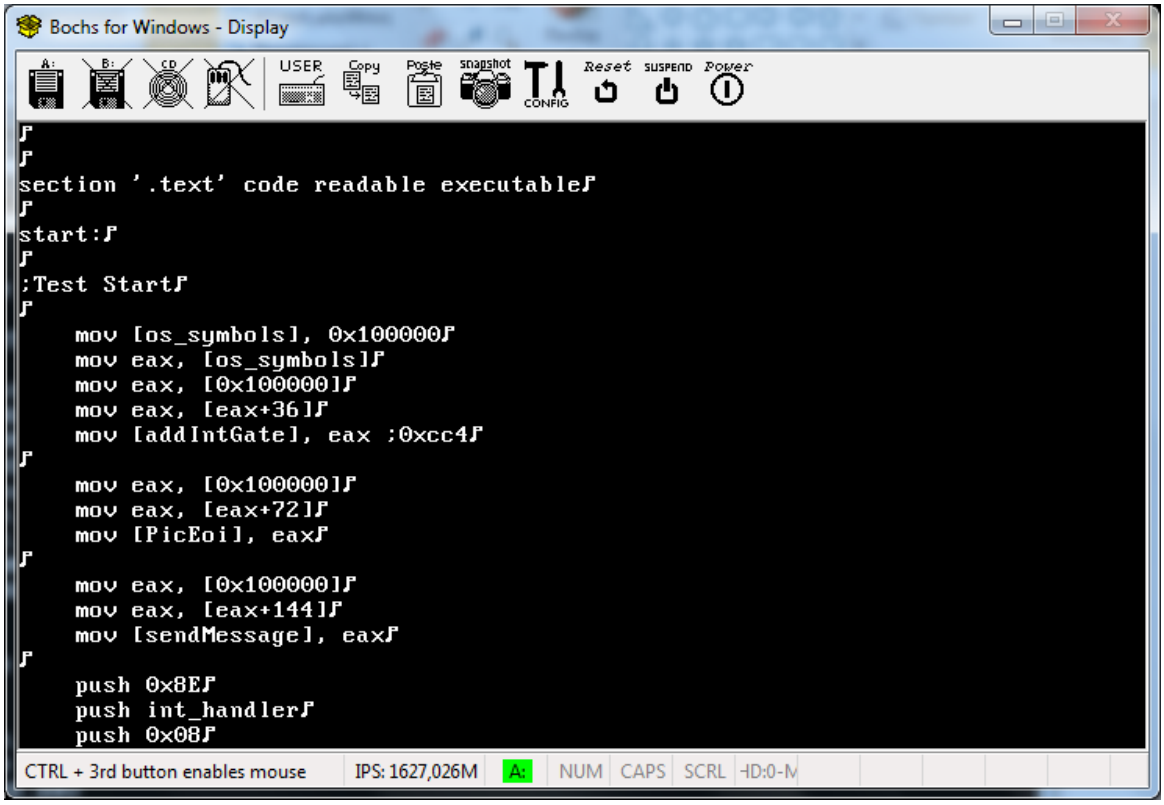
Όπως αναφέρθηκε και προηγουμένως εκτός των εφαρμογών χρήστη που αποτελούν μέρος του λειτουργικο,ύ συμπεριλήφθησαν και κάποιες εφαρμογές οι οποίες δεν είναι μέρος του αλλά ήταν χρήσιμες για τον σκοπό της παρουσίασης καθώς και για να παρέχουν μία πιο ολοκληρωμένη εικόνα του συστήματος. Οι εφαρμογές αυτές ήταν χρήσιμες και κατά την φάση της ανάπτυξης για δοκιμαστικούς σκοπούς αλλά και για ελέγξουμε στην πράξη πως αλληλεπιδρά ο πυρήνας με διάφορες εφαρμογές χρήστη.

Η πρώτη εφαρμογή αφορά έναν κειμενογράφο. Μπορούμε να τον καλέσουμε από την γραμμή εντολών γράφοντας το όνομα του εκτελέσιμου με όρισμα το όνομα ενός αρχείου. Ο κειμενογράφος ελέγχει εάν το αρχείο υπάρχει και εμφανίζει το περιεχόμενο του στην οθόνη. Αρκετές κλήσεις συστήματος εξυπηρετούνται από τον πυρήνα κατά την φάση αυτή για άνοιγμα αρχείου, ανάγνωση, δυναμική δέσμευση μνήμης και έξοδο στην οθόνη του τερματικού. Έπειτα, η διεργασία μπλοκάρεται για να μην δεσμεύει πόρους μέχρι ένα νέο μήνυμα να έρθει στην ουρά. Αυτός ο μηχανισμός είναι παρόμοιος με την ουρά μηνυμάτων του WindowsApi. Μόλις ο χρήστης της εφαρμογής

πατήσει κάποιο πλήκτρο η διεργασία ξεμπλοκάρεται και διαβάζει την είσοδο του χρήστη. Ο χρήστης μπορεί να πλοηγηθεί στο κείμενο με τα βελάκια του πληκτρολογίου η με τα πλήκτρα *home, endpageup*και*pagedown*. Η συμπεριφορά τους είναι παρόμοια με αυτή των περισσότερων κειμενογράφων. Αν για παράδειγμα πατήσουμε το κουμπί *home* ενώ ο κέρσορας βρίσκεται στον πρώτο μη κενό χαρακτήρα της γραμμής τότε ο κέρσορας μετακινείται στην αρχή της γραμμής. Αν όμως ο κέρσορας βρίσκεται σε οποιοδήποτε άλλο σημείο της γραμμής τότε μετακινείται στον πρώτο χαρακτήρα της γραμμής. Τα πλήκτρα *backspace* και *delete* διαγράφουν τον αμέσως προηγούμενο ή επόμενο χαρακτήρα από τη θέση του κέρσορα αντίστοιχα, ενώ τα πλήκτρα εκτυπώσιμων χαρακτήρων εισάγουν τον αντίστοιχο χαρακτήρα στη θέση του κέρσορα. Αν το πλήκτρο *shift* είναι πατημένο, γίνεται εισαγωγή του αντίστοιχου χαρακτήρα. Τέλος με το πλήκτρο *f8* γίνεται αποθήκευση του αρχείου και με το πλήκτρο *Esc* έξοδος και τερματισμός του προγράμματος. Ο κειμενογράφος έχει υλοποιηθεί με τη δομή δεδομένων gapbuffer

Το δεύτερο σημαντικό πρόγραμμα που συμπεριλήφθηκε είναι το Fasm[28]Πρόκειται για μία εφαρμογή ανοιχτού κώδικα γραμμένη σε συμβολική γλώσσα από τον Tomasz Grysztar.Το Fasmείναι ένας συμβολομεταφραστής. Ο κώδικας της εφαρμογής χρησιμοποιήθηκε σχεδόν αυτούσιος με μικρές αλλά σημαντικές αλλαγές, ώστε να μπορεί να εκτελεστεί πάνω σε αυτό το λειτουργικό (porting).

Ο κειμενογράφος σε συνδυασμό με τον συμβολομεταφραστή FASMδίνει την δυνατότητα στον χρήστη να γράψει δικά του προγράμματα και οδηγούς συσκευών και να τα μεταγλωττίσει πάνω σε αυτό το λειτουργικό προσφέροντας αυτονομία.



Εικόνα 3. Αρχείο με κώδικα σε συμβολική γλώσσα στον κειμενογράφο

2.2.3 Ένα ολοκληρωμένο παράδειγμα

Σε αυτή την υποενότητα γίνεται μία εκτενέστερη παρουσίαση της εφαρμογής από την πλευρά του χρήστη και του προγραμματιστή εφαρμογών μέσω ενός ολοκληρωμένου παραδείγματος. Σε αυτό το παράδειγμα θα γράψουμε και θα μεταγλωττίσουμε έναν οδηγό συσκευής για το ποντίκι του υπολογιστή.

Αρχικά, με την χρήση του κειμενογράφου θα συντάξουμε το πρόγραμμα σε συμβολική γλώσσα. Στην συνέχεια με τον συμβολομεταφραστήFasmθα μεταγλωττίσουμε τον οδηγό συσκευής. Ο οδηγός θα διαβάζει την είσοδο από το ποντίκι και θα στέλνει μηνύματα στο λειτουργικό. Το λειτουργικό με τη σειρά του θα ανακατευθύνει τα μηνύματα αυτά στις εκτελούμενες διεργασίες. Τέλος θα δημιουργηθεί μία εφαρμογή που θα διαβάζει τα μηνύματα αυτά και θα εμφανίζει τις συντεταγμένες του κέρσορα. Η εφαρμογή αυτή θα μπορούσε επίσης να γραφτεί σε συμβολική γλώσσα με το Fasm. Ωστόσο για να είναι πιο πλήρης η παρουσίαση θα γραφτεί σε Ckαι θα μεταγλωττιστεί με το GCCστα Windowsσε μορφή κατάλληλη ώστε να τρέξει στο παρον λειτουργικό (crosscompile).

Το πρώτο βήμα είναι να ανοίξουμε τον κειμενογράφο με την εντολή *NOTEPAD.RUNA:/MOUSE.ASM* από τη γραμμή εντολών. Ως όρισμα δηλαδή περνάμε την διαδρομή ενός κενού αρχείου. Κατά την εκτέλεση του ο κειμενογράφος μας εμφανίζει ένα δικό του παράθυρο στο οποίο μπορούμε να επεξεργαστούμε κείμενο. Με το

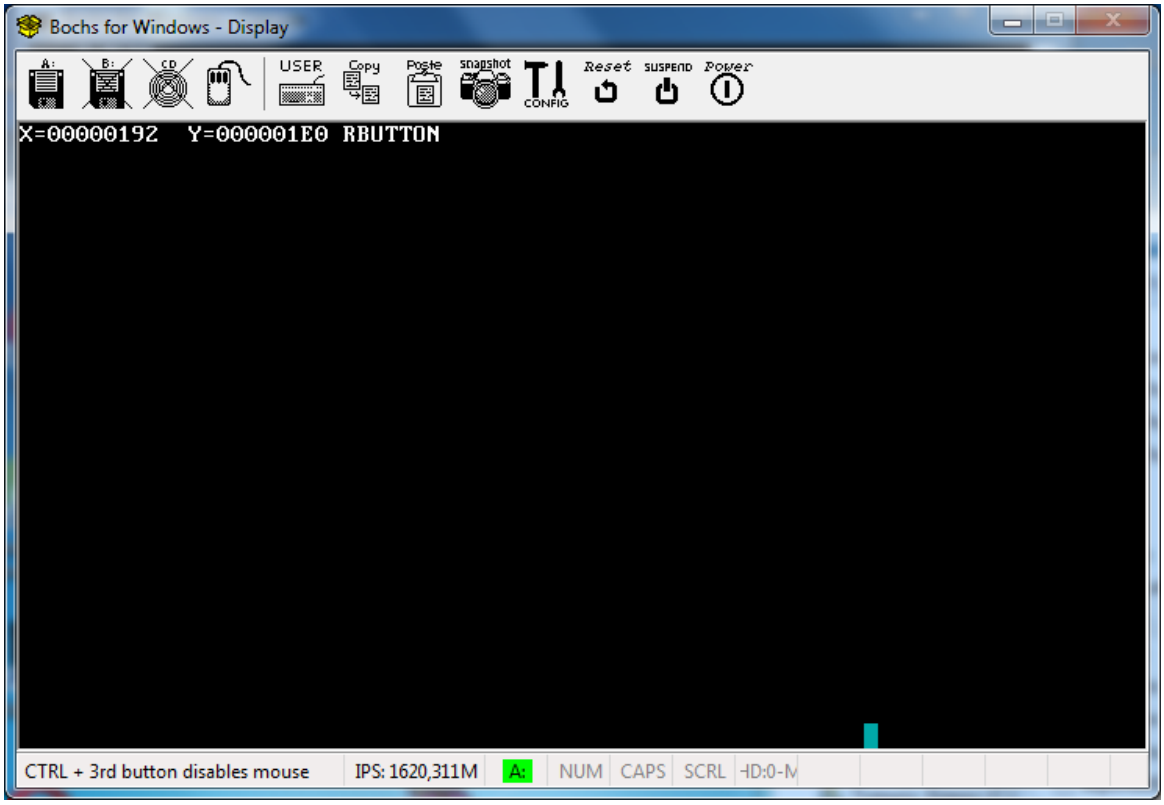
πλήκτρο *f11* μπορούμε να εναλλάσσουμε το παράθυρο που είναι στο προσκήνιο εφόσον έχουμε περισσότερες από μία εφαρμογές ανοιχτές.

Έπειτα στον κειμενογράφο συντάσσουμε το πρόγραμμα σε συμβολική γλώσσα. Το πρόγραμμα αυτό διαβάζει διευθύνσεις μνήμης στις οποίες στέλνει δεδομένα στο ποντίκι. Μόλις ανιχνεύσει κίνηση στέλνει ένα μήνυμα στην ουρά μηνυμάτων του λειτουργικού. Το λειτουργικό από εκεί πέρα θα αποφασίσει σε ποια ή σε ποιες διεργασίες θα στείλει το μήνυμα. Ο κώδικας της εφαρμογής βρίσκεται στο αρχείο *MOUSE.ASM.*

Μόλις ολοκληρώσουμε την σύνταξη του προγράμματος αποθηκεύουμε με το *f8* και με το πλήκτρο *esc* τερματίζουμε την εκτέλεση του κειμενογράφου. Με την εντολή*FASM.RUNA:/MOUSE.ASMA:/MOUSE.DRV* μεταγλωττίζουμε το πρόγραμμα. Το πρώτο όρισμα είναι η διαδρομή του πηγαίου αρχείου και το δεύτερο το όνομα του παραγόμενου εκτελέσιμου αρχείου. Αν δεν υπάρχει κάποιο σφάλμα στον κώδικα τότε στο τέλος της διαδικασίας θα εμφανιστεί μήνυμα επιτυχίας ενώ διαφορετικά θα εμφσνιστεί κάποιο μήνυμα σφάλματος. Σε περίπτωση σφάλματος θα πρέπει να ελέγξουμε τον πηγαίο κώδικα και να διορθώσουμε τυχόν συντακτικά λάθη.

Εάν η μεταγλώττιση ολοκληρωθεί με επιτυχία το επόμενο βήμα θα είναι να εκκινήσουμε τον οδηγό συσκευής. Υπάρχουν δύο τρόποι για να ξεκινήσει ο οδηγός συσκευής. Ο πρώτος τρόπος θα ήταν να με την εντολή *drvload.* Δέχεται ως όρισμα την θέση του αρχείου του οδηγού συσκευής. Το μειονέκτημα είναι ότι με αυτόν τον τρόπο θα έπρεπε κάθε φορά να εκκινούμε τον οδηγό συσκευής. Επομένως αυτός ο τρόπος είναι χρήσιμος μόνο για δοκιμαστικούς σκοπούς. Ο δεύτερος τρόπος θα ήταν να καταχωρήσουμε μία εγγραφή στο αρχείο *DRIVERS.CAT* με τη διαδρομή προς τον οδηγό συσκευής. Κατά την εκκίνηση του λειτουργικού μία υπηρεσία εκτελείται η οποία ελέγχει αυτό το αρχείο για εγγραφές. Για κάθε εγγραφή που βρίσκει φορτώνει και τον αντίστοιχο οδηγό συσκευής. Έτσι με αυτόν τον τρόπο ο οδηγός συσκευής θα εκτελείται συνέχεια χωρίς να χρειάζεται επέμβαση από τον χρήστη κάθε φορά. Φυσικά μετά την καταχώρηση τις εγγραφής στο σύστημα θα πρέπει να γίνει επανεκκίνηση για να φορτωθεί καθώς πλέον θα φορτώνεται με την εκκίνηση του υπολογιστή.

Το τελευταίο βήμα θα είναι να γράψουμε ένα πρόγραμμα το οποίο θα διαβάσει από τον οδηγό συσκευών την κίνηση του ποντικιού στους δύο άξονες και θα εμφανίζει τη θέση στην οποία θα εμφανιζόταν ο δείκτης του ποντικιού αν υπήρχε. Το πρόγραμμα αυτό αποτελεί μέρος της εργασίας και ο πηγαίος κώδικας της εφαρμογής αυτής βρίσκεται στον φάκελο *Os_mouseTest.*



Εικόνα 4. Το πρόγραμμα εμφανίζει τις συντεταγμένες του κέρσορα στην οθόνη.

Συμπεράσματα

Για τη σχεδίαση και υλοποίηση του πυρήνα είναι αναγκαία η μελέτη, η σχεδίαση και η υλοποίηση των πολλών διαφορετικών υποσυστημάτων του. Κάποια από αυτά τα υποσυστήματα είναι απλούστερα ενώ άλλα είναι ιδιαίτερα πολύπλοκα. Παρά την πολυπλοκότητα τους όμως, είναι εφικτή η υλοποίηση λειτουργικών υποσυστημάτων που δεν περιέχουν λογικά σφάλματα ή κενά ασφαλείας με ιδιαίτερη προσοχή. Το μεγαλύτερο πρόβλημα ωστόσο είναι η αλληλεπίδραση μεταξύ αυτών των υποσυστημάτων. Για παράδειγμα, σχεδόν όλα τα υποσυστήματα έχουν την ανάγκη να χρησιμοποιούν τη μνήμη του υπολογιστή μέσω του διαχειριστή μνήμης. Ταυτόχρονα, ο διαχειριστής μνήμης μπορεί να θελήσει να αποθηκεύσει μόνιμα δεδομένα μέσω του συστήματος αρχείων. Το σύστημα αρχείων με την σειρά του χρησιμοποιεί τον διαχειριστή μνήμης. Αυτή η αλληλεπίδραση μεταξύ των διαφορετικώνυποσυστημάτων είναι που κάνει αρκετά δύσκολή τη διεκπεραίωση του συνολικού έργου.

Επομένως, το γενικότερο συμπέρασμα κατά την εκπόνηση της εργασίας είναι ότι το μέγεθος ενός έργου δεν σχετίζεται γραμμικά με τον απαιτούμενο χρόνο διεκπεραίωσης του. Όσο αυξάνεται η πολυπλοκότητα απαιτείται δυσανάλογα περισσότερος χρόνος για μελέτη, σχεδίαση και υλοποίηση αφού ακόμα και μία μικρή αλλαγή ή προσθήκη στο έργο μπορεί να επηρεάζει πολλά διαφορετικά μέρη του.

Ειδικότερα για τη σχεδίαση του πυρήνα συμπεραίνουμε ότι θα πρέπει να υλοποιεί μόνο τις απαραίτητες λειτουργίες που δεν μπορούν να υλοποιηθούν σε υψηλότερο επίπεδο. Οι υπόλοιπες λειτουργίες θα πρέπει να υλοποιούνται από προγράμματα ή βιβλιοθήκες στον χώρο του χρήστη. Αυτά τα προγράμματα είναι στην ουσία επέκταση του λειτουργικού ωστόσο καθώς εκτελούνται στον χώρο του χρήστη τυχόν αποτυχία τους δεν θα επηρεάσει ολόκληρο το σύστημα κάνοντας το έτσι περισσότερο ανθεκτικό στα διάφορα σφάλματα που μπορεί να παρουσιαστούν.

Πολλές επιπλέον δυνατότητες θα μπορούσαν να προστεθούν σε μελλοντικές εκδόσεις, επεκτείνοντας τη λειτουργικότητα του πυρήνα. Υπάρχουν κάποιες βασικές λειτουργίες οι οποίες θα μπορούσαν να συμπεριληφθούν και πάρα πολλές δευτερεύουσες.

Η πρώτη βασική λειτουργία είναι η σελιδοποίηση και ο εικονικός χώρος διευθύνσεων. Οι οικογένεια επεξεργαστών x86 υποστηρίζει αυτές τις δυνατότητες. Μία ακόμα σημαντική προσθήκη που θα μπορούσε να γίνει είναι η υποστήριξη πολυνηματικών εφαρμογών. Μαζί με την υποστήριξη πολυνηματικών εφαρμογών θα ήταν πολύ χρήσιμη και δυνατότητα υποστήριξης πολλαπλών επεξεργαστών ή πολυπύρηνων τσιπ. Όλοι οι σύγχρονοι επεξεργαστές τις οικογένειαςx86 είναι πολυπύρηνοι. Ωστόσο το παρόν λειτουργικό αξιοποιεί μόνο έναν από τους διαθέσιμους πυρήνες.

Άλλη μία βασική προσθήκη θα ήταν η υλοποίηση γραφικού περιβάλλοντος χρήστη. Η εφαρμογή έχει σχεδιαστεί έτσι ώστε στο μέλλον να μπορεί να υποστηρίξει γραφικό περιβάλλον με ελάχιστες αλλαγές.

Υπάρχουν φυσικά και πολλές άλλες προσθήκες που θα μπορούσαν να υλοποιηθούν όπως για παράδειγμα χρήστες, δικαιώματα χρήστη, κρυπτογράφηση δίσκου, διαχειριστής διεργασιών και υποστήριξηUEFI κατά την εκκίνηση.

Βιβλιογραφία

[1] Karl-Bridge-Microsoft et al., «PE Format,» docs.microsoft.com, 23 June 2022. [Ηλεκτρονικό]. Available: https://docs.microsoft.com/en-us/windows/win32/debug/pe-format#machine-types. [Πρόσβαση 22 August 2022].

[2] Gerald Pfeifer et al., "x86 Function Attribute," https://gcc.gnu.org, 14 November 2019. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc-7.5.0/gcc/x86-Function-Attributes.html#x86-Function-Attributes. [Accessed 20 August 2022].

[3] Richard M. Stallman et al, «Status of Supported Architectures from Maintainers' Point of View,» https://gcc.gnu.org, 28 July 2021. [Ηλεκτρονικό]. Available: https://gcc.gnu.org/onlinedocs/gcc/x86-Options.html. [Πρόσβαση 24 August 2022].

[4] Simon Tatham, et al, «nasm,» nasm.us, [Ηλεκτρονικό]. Available: https://www.nasm.us. [Πρόσβαση 3 September 2022].

[5] 8bittree, «Why does BIOS operate in 16-bits instead of 32/64-bits?,» superuser.com, 18 October 2022. [Ηλεκτρονικό]. Available: https://superuser.com/questions/988308/why-does-bios-operate-in-16-bits-instead-of-32-64-bits. [Πρόσβαση 13 September 2022].

[6] A. S. Mutschler, «Debugging Embedded Applications,» semiengineering.com, 10 November 2021. [Ηλεκτρονικό]. Available: https://semiengineering.com/debugging-embedded-applications/. [Πρόσβαση 24 August 2022].

[7] Tim Butler,et al, «Bochs,» bochs.sourceforge.io, [Ηλεκτρονικό]. Available: https://bochs.sourceforge.io. [Πρόσβαση 8 September 2022].

[8] HHD Software Ltd, «Free Hex Editor Neo,» HHD Software Ltd, [Ηλεκτρονικό]. Available: https://www.hhdsoftware.com/free-hex-editor. [Πρόσβαση 29 August 2022].

[9] w77, «ImDisk Toolkit,» [Ηλεκτρονικό]. Available: https://sourceforge.net/projects/imdisk-toolkit/. [Πρόσβαση 26 August 2022].

[10] Richard M. Stallman et al., «Option Summary,» gcc.gnu.org, [Ηλεκτρονικό]. Available: https://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html. [Πρόσβαση 24 Ausugst 2022].

[11] M. Petch, «Linking a file using ld to output a binary file gives error in OS developmen,» stackoverflow.com, 20 May 2019. [Ηλεκτρονικό]. Available: https://stackoverflow.com/questions/37344575/linking-a-file-using-ld-to-output-a-binary-file-gives-error-in-os-development. [Πρόσβαση 28 August 2022].

[12] Sandra A. Moore, et al., «Red Hat Enterprise Linux 4: Reference Guide Chapter 1. Boot Process, Init, and Shutdown,» web.mit.edu, [Ηλεκτρονικό]. Available:

https://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-en-4/s1-boot-init-shutdown-process.html. [Πρόσβαση 30 August 2022].

[13] Jhawthorn,et al., «System Initialization (x86),» wiki.osdev.org, 23 November 2020. [Ηλεκτρονικό]. Available: https://wiki.osdev.org/System_Initialization_(x86). [Πρόσβαση 30 August 2022].

[14] R. D. Brow, «The x86 Interrupt List,» cs.cmu.edu, 28 February 2022. [Ηλεκτρονικό]. Available: http://www.cs.cmu.edu/afs/cs.cmu.edu/user/ralf/pub/WWW/files.html. [Πρόσβαση 30 August 2022].

[15] Anog,et al., «A20 Line,» wiki.osdev.org, 13 January 2007. [Ηλεκτρονικό]. Available: https://wiki.osdev.org/A20_Line. [Πρόσβαση 30 August 2022].

[16] Intel® 64 and IA-32 Architectures Software Developer's Manual, Intel, pp. 3-6 - 3-10, 01 April 2022. [Ηλεκτρονικό]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html. [Πρόσβαση 3 September 2022].

[17] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3, Intel pp. 2-1 - 2-4, 2-12, [Ηλεκτρονικό]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html. [Πρόσβαση 03 September 2022].

[18] A. S. Tanenbaum, «Διαχείριση μνήμης,» σε *Σύγχρονα λειτουργικά συστήματα*, 3rd επιμ., Άμστερνταμ, Κλειδάριθμος, 2008, pp. 227-309.

[19] Intel, «Intel® 64 and IA-32 Architectures Software Developer's Manual pp. 3-5,» [Ηλεκτρονικό]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html. [Πρόσβαση 5 September 2022].

[20] Abraham Silberschatz,Peter Baer Galvin,Greg Gagne, Λειτουργικά Συστήματα, 2nd επιμ., New Haven: Ίων, 2005, p. 385.

[21] Abraham Silberschatz,Peter Baer Galvin,Greg Gagne, Λειτουργικά συστήματα, 2nd επιμ., New Haven: Ίων, 2005, pp. 661-666.

[22] Intel, «Calls to Other Privilege Levels pp.6-7 6-8,» [Ηλεκτρονικό]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html. [Πρόσβαση 12 September 2022].

[23] A. IWAYA, «Why Do x86 CPUs Only Use Two Out of Four "Rings"?,» howtogeek.com, 22 April 2016. [Ηλεκτρονικό]. Available: https://www.howtogeek.com/251081/why-do-x86-cpus-only-use-two-out-of-four-rings/. [Πρόσβαση 10 September 2022].

[24] Jamie Hanrahan, «Why do x86 CPUs only use 2 out of 4 rings?,» superuser.com, 12 April 2016. [Ηλεκτρονικό]. Available: https://superuser.com/questions/1063420/why-

do-x86-cpus-only-use-2-out-of-4-rings. [Πρόσβαση 10 September 2022].

[25] Ineo, et al., «System Calls,» wiki.osdev.org, 11 March 2007. [Ηλεκτρονικό]. Available: https://wiki.osdev.org/System_Calls. [Πρόσβαση 12 September 2022].

[26] GrantMeStrength, et al., «API Index for desktop Windows applications,» learn.microsoft.com, 28 April 2021. [Ηλεκτρονικό]. Available: https://learn.microsoft.com/en-us/windows/win32/apiindex/api-index-portal. [Πρόσβαση 14 September 2022].

[27] Lahwaacz, et al., «Kernel module,» wiki.archlinux.org, 23 September 2015. [Ηλεκτρονικό]. Available: https://wiki.archlinux.org/title/Kernel_module. [Πρόσβαση 18 September 2022].

[28] T. Grysztar, «flat assembler,» flatassembler.net, [Ηλεκτρονικό]. Available: https://flatassembler.net. [Πρόσβαση 18 September 2022].

Παράρτημα Κώδικα

Αρχείο: 64bitDivision.c

https://www.cs.usfca.edu/~benson/cs326/pintos/pintos/src/lib/arithmetic.c

https://www.cs.usfca.edu/~benson/cs326/pintos/pintos/LICENSE

Copyright (C) 2004, 2005, 2006 Board of Trustees, Leland Stanford Jr. University. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A few individual files in Pintos were originally derived from other projects, but they have been extensively modified for use in Pintos. The original code falls under the original license, and modifications

for Pintos are additionally covered by the Pintos license above.

In particular, code derived from Nachos is subject to the following
license:

```
/* Copyright (c) 1992-1996 The Regents of the University of California.
   All rights reserved.

   Permission to use, copy, modify, and distribute this software
   and its documentation for any purpose, without fee, and
   without written agreement is hereby granted, provided that the
   above copyright notice and the following two paragraphs appear
   in all copies of this software.

   IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO
   ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR
   CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE
   AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA
   HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

   THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY
   WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
   WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
   PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS"
   BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
   PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR
   MODIFICATIONS.
*/
#include "type.h"

/* On x86, division of one 64-bit integer by another cannot be
   done with a single instruction or a short sequence.  Thus, GCC
```

```
   implements 64-bit division and remainder operations through
   function calls.  These functions are normally obtained from
   libgcc, which is automatically included by GCC in any link
   that it does.

   Some x86-64 machines, however, have a compiler and utilities
   that can generate 32-bit x86 code without having any of the
   necessary libraries, including libgcc.  Thus, we can make
   Pintos work on these machines by simply implementing our own
   64-bit division routines, which are the only routines from
   libgcc that Pintos requires.

   Completeness is another reason to include these routines.  If
   Pintos is completely self-contained, then that makes it that
   much less mysterious. */

/* Uses x86 DIVL instruction to divide 64-bit N by 32-bit D to
   yield a 32-bit quotient.  Returns the quotient.
   Traps with a divide error (#DE) if the quotient does not fit
   in 32 bits. */
static inline uint32_t
divl (uint64_t n, uint32_t d)
{
  uint32_t n1 = n >> 32;
  uint32_t n0 = n;
  uint32_t q, r;

  __asm ("divl %4"
      : "=d" (r), "=a" (q)
      : "0" (n1), "1" (n0), "rm" (d));

  return q;
```

```
}

/* Returns the number of leading zero bits in X,
   which must be nonzero. */
static int
nlz (uint32_t x)
{
  /* This technique is portable, but there are better ways to do
     it on particular systems.  With sufficiently new enough GCC,
     you can use __builtin_clz() to take advantage of GCC's
     knowledge of how to do it.  Or you can use the x86 BSR
     instruction directly. */
  int n = 0;
  if (x <= 0x0000FFFF)
    {
      n += 16;
      x <<= 16;
    }
  if (x <= 0x00FFFFFF)
    {
      n += 8;
      x <<= 8;
    }
  if (x <= 0x0FFFFFFF)
    {
      n += 4;
      x <<= 4;
    }
  if (x <= 0x3FFFFFFF)
    {
      n += 2;
      x <<= 2;
```

```
   }
  if (x <= 0x7FFFFFFF)
    n++;
  return n;
}

/* Divides unsigned 64-bit N by unsigned 64-bit D and returns the
   quotient. */
static uint64_t
udiv64 (uint64_t n, uint64_t d)
{
  if ((d >> 32) == 0)
    {
      /* Proof of correctness:

         Let n, d, b, n1, and n0 be defined as in this function.
         Let [x] be the "floor" of x.  Let T = b[n1/d].  Assume d
         nonzero.  Then:
            [n/d] = [n/d] - T + T
                  = [n/d - T] + T                    by (1) below
                  = [(b*n1 + n0)/d - T] + T          by definition of n
                  = [(b*n1 + n0)/d - dT/d] + T
                  = [(b(n1 - d[n1/d]) + n0)/d] + T
                  = [(b[n1 % d] + n0)/d] + T,        by definition of %
         which is the expression calculated below.

         (1) Note that for any real x, integer i: [x] + i = [x + i].

         To prevent divl() from trapping, [(b[n1 % d] + n0)/d] must
         be less than b.  Assume that [n1 % d] and n0 take their
         respective maximum values of d - 1 and b - 1:
               [(b(d - 1) + (b - 1))/d] < b
```

```
          <=> [(bd - 1)/d] < b
          <=> [b - 1/d] < b
        which is a tautology.

        Therefore, this code is correct and will not trap. */
     uint64_t b = 1ULL << 32;
     uint32_t n1 = n >> 32;
     uint32_t n0 = n;
     uint32_t d0 = d;

     return divl (b * (n1 % d0) + n0, d0) + b * (n1 / d0);
   }
  else
   {
     /* Based on the algorithm and proof available from
        http://www.hackersdelight.org/revisions.pdf. */
     if (n < d)
       return 0;
     else
       {
         uint32_t d1 = d >> 32;
         int s = nlz (d1);
         uint64_t q = divl (n >> 1, (d << s) >> 32) >> (31 - s);
         return n - (q - 1) * d < d ? q - 1 : q;
       }
   }
}

/* Divides unsigned 64-bit N by unsigned 64-bit D and returns the
   remainder. */
static uint32_t
umod64 (uint64_t n, uint64_t d)
```

```
{
  return n - d * udiv64 (n, d);
}

/* Divides signed 64-bit N by signed 64-bit D and returns the
   quotient. */
static int64_t
sdiv64 (int64_t n, int64_t d)
{
  uint64_t n_abs = n >= 0 ? (uint64_t) n : -(uint64_t) n;
  uint64_t d_abs = d >= 0 ? (uint64_t) d : -(uint64_t) d;
  uint64_t q_abs = udiv64 (n_abs, d_abs);
  return (n < 0) == (d < 0) ? (int64_t) q_abs : -(int64_t) q_abs;
}

/* Divides signed 64-bit N by signed 64-bit D and returns the
   remainder. */
static int32_t
smod64 (int64_t n, int64_t d)
{
  return n - d * sdiv64 (n, d);
}

/* These are the routines that GCC calls. */

long long __divdi3 (long long n, long long d);
long long __moddi3 (long long n, long long d);
unsigned long long __udivdi3 (unsigned long long n, unsigned long long d);
unsigned long long __umoddi3 (unsigned long long n, unsigned long long d);

/* Signed 64-bit division. */
long long
```

```
__divdi3 (long long n, long long d)
{
  return sdiv64 (n, d);
}

/* Signed 64-bit remainder. */
long long
__moddi3 (long long n, long long d)
{
  return smod64 (n, d);
}

/* Unsigned 64-bit division. */
unsigned long long
__udivdi3 (unsigned long long n, unsigned long long d)
{
  return udiv64 (n, d);
}

/* Unsigned 64-bit remainder. */
unsigned long long
__umoddi3 (unsigned long long n, unsigned long long d)
{
  return umod64 (n, d);
}
```

Αρχείο: contexSwitch.asm

```
[BITS 32]

global contexSwitch

global _emptyStack

extern _shedule
extern _saveKrnlStck ;save esp of the running procces in its pcb(procces  code bkock)
extern _loadKrnlStck
EXTERN _PicEoi

%define PIT_IRQ    0


extern _sysWait  ;if no procces is ready then iret and sysWait


contexSwitch:
            pop dword [returnAdd] ;move return address out of the stack before change stack
            jmp saveState
shedule:       call _shedule
            jmp loadState
return:        push dword [returnAdd]

            ret



saveState:
            pushad
```

```
        push esp
        call _saveKrnlStck
        add esp, 0x4

        jmp shedule



loadState:
        call _loadKrnlStck
        cmp eax, 0
        jne startNextTask

        ;this works
        ;push PIT_IRQ
        ;call _PicEoi
        ;add esp, 0x18
        ;jmp _sysWait


        ;mov esp, emptyStack+32
        ;popad  ;??
        ;push ss
        ;push dword _emptyStack+32
        mov esp, _emptyStack+1020
        pushfd
        push 0x8
        push _sysWait
        ;push cs
        ;push _sysWait
        jmp return
```

```
startNextTask:  mov esp, eax
            popad

            jmp return



returnAdd:      dd 0x00
_emptyStack:    TIMES 256 DD 0x00
```

Αρχείο: createProcces.c

```
#include "createProcces.h"
#include "memory.h"
#include "stdiok.h"
#include "kernelHeap.h"
#include "file.h"
#include "pe.h"
#include "ProcCondition.h"
#include "gui.h"
#include "pic.h"

int kkk;

extern pcbNode *readyProccesQueue, *currentProcces, *readyProccesQueueTail,
*blockedProccesQueue;  /**??exter in header*/

cond_t cond_emtpyQueue; ///??

static pcbNode* newPcb(void)
{
   pcbNode *pcb = NULL;

   if( (pcb = kmalloc(sizeof(pcbNode)) ) == NULL)
      return NULL;

   if(readyProccesQueue == NULL)
      readyProccesQueue = readyProccesQueueTail = pcb;
   else
   {
      readyProccesQueueTail->next = pcb;
      readyProccesQueueTail = readyProccesQueueTail->next;
   }
```

```
    readyProccesQueueTail->next = NULL;


    return pcb;


}


static dword loadProcces(const char *exepath, proc_mem_regions *reg, pcbNode *newProc,
const char *params)
{
    dword entry = 0;
    pcbNode *temp = currentProcces;
    currentProcces = newProc; /*set temporarily new procces as current procces to make memory
allocation calls*/
    const size_t stackSz = 10, kstackSz = 10;
    byte* procParams = NULL;
    int i, j;



    /*Improve: createProccesCatalog take as argument a pcbNode */
    newProc->pageCat.count = 1;
    if((newProc->pageCat.catalog = createProccessCatalog()) == NULL)
        return 1;


    loadProcFromDisk(exepath, &entry, 0);
    reg->entry = (void*) entry;


    reg->stack = (byte*) ScMemAlloc(stackSz, 0, ALLOC_MODE_ANY_ADD, 0);
    if(reg->stack == NULL)
        return 1;




    procParams = (byte *) ScMemAlloc(1, 0, ALLOC_MODE_ANY_ADD, 0);
```

```
   if(procParams  == NULL)
      return 1;

   //ScMemAlloc(1, 0, ALLOC_MODE_ANY_ADD, 0);

   memcpy(params, procParams, strlen(params) );
   procParams[strlen(params)] = '\0';

   *((dword*) ( reg->stack + ((stackSz * PAGE_FRAME_SIZE) - sizeof(dword) ))) =  procParams;

   reg->stacksize = stackSz;
   //reg->stack += (stackSz * PAGE_FRAME_SIZE) - sizeof(dword); /*esp initialized at the */

   reg->kstack = (byte*) ScMemAlloc(kstackSz, 0, ALLOC_MODE_ANY_ADD, 0);
   if(reg->kstack == NULL)
      return 1;


   //printk("procparams %x\n", (dword) reg->stack + ((reg->stacksize * PAGE_FRAME_SIZE) ) );
   //for(i = 0;  i <20000000; i++);    ///??? Error when delay

   //kkk++;
   //if(kkk == 3){ __asm volatile("cli;"); printk("%x", reg->stack); while(1);}

   reg->kstacksize = kstackSz;

   currentProcces = temp;

   return 0;
}

#define KERNEL_SPACE_EFALGS 0x216
```

```
#define KERNEL_CODE_SEGMENT 0x08
#define USER_CODE_SEGMENT 0x23
#define USER_DATA_SEGMENT 0x1B
static void initStack(proc_mem_regions *reg, pcbNode *newProc)
{
   /*initialize stacks and set esp and kernelEsp in pcbNode*/
   newProc->esp0 = (dword) (reg->kstack + ((reg->kstacksize * PAGE_FRAME_SIZE) -
sizeof(dword)));/*esp initialized at the bottom of the stack*/
   newProc->esp =  newProc->esp0 - 52 + 4; /*iret + pushad bytes*/

   (*((dword*)newProc->esp0 )) = USER_DATA_SEGMENT;
   (*((dword*)(newProc->esp0 - 4) )) = (dword) reg->stack + ((reg->stacksize *
PAGE_FRAME_SIZE) - sizeof(dword) - 4 );
   (*((dword*)(newProc->esp0 - 8) )) = KERNEL_SPACE_EFALGS;
   (*((dword*)(newProc->esp0 - 12) )) = USER_CODE_SEGMENT;
   (*((dword*)(newProc->esp0 - 16) )) = (dword) reg->entry;

   //printk("esp: %x\n", newProc->esp);
   //printk("esp0: %x\n", newProc->esp0);
   //while(1);

}

/*add a child node to the current procces's list*/
static void pcbAddChild(pcbNode *newProc)
{
   childProccesNode *newChildNode = NULL;

   if(currentProcces != NULL)
   {
      if(currentProcces->childPidListHead == NULL)
      {
```

```
            currentProcces->childPidListHead = kmalloc(sizeof(childProccesNode));   /*Not checking if kmallod fails*/
            currentProcces->childPidListHead->next = NULL;
        }
        else{
            newChildNode = kmalloc(sizeof(childProccesNode));
            newChildNode->next = currentProcces->childPidListHead;
            currentProcces->childPidListHead = newChildNode;
        }

        currentProcces->childPidListHead->hasReturn = false;
        currentProcces->childPidListHead->pid = newProc->pid;
        currentProcces->childPidListHead->pcb = newProc;
    }
}

static dword createPid(void)
{
    static dword pid = 1;

    return pid++;
}

static void initPcb(pcbNode *newProc)
{
    newProc->state = ready;
    newProc->pid = createPid();
    newProc->pType = P_TYPE_BACKGROUND;
    newProc->videoBuffer = NULL;


    init_condition(&newProc->msgQueue.cond_emtpyQueue);  ///???
```

```
    newProc->msgQueue.first = newProc->msgQueue.last = 0;

    if(currentProcces != NULL)
    {
        newProc->parentPcb = currentProcces;
        newProc->parentPid = currentProcces->pid;
        pcbAddChild(newProc);
    }
    else{
        newProc->parentPcb = NULL;
        newProc->parentPid = 0;
    }

    newProc->childPidListHead = NULL; /*new procces has no childs yet*/
}

dword removeMessageFromProc(pcbNode *proc, proccesMsg *msg)
{
    dword ret, first = (proc->msgQueue.first + 1) % MSG_QUEUE_SIZE;

    if( proc->msgQueue.first == proc->msgQueue.last )
        return 0xFFFFFFFF;  /**queue is empty*/
__asm volatile("cli;");
    ret = msg->msg = proc->msgQueue.procMsg[first].msg;
    msg->param = proc->msgQueue.procMsg[first].param;
    msg->msgType = proc->msgQueue.procMsg[first].msgType;
    proc->msgQueue.first = first;

__asm volatile("sti;");
    return ret;
}
```

```
dword ScgetMessage(proccesMsg *msg)  /**??File.c*/
{
   dword ret;

   //__asm volatile("cli;");
   while( (ret = removeMessageFromProc(currentProcces,  msg)) == 0xFFFFFFFF
)condWait(&currentProcces->msgQueue.cond_emtpyQueue); /**??block Proccess*/
   //__asm volatile("sti;");


   return ret;
}

bool addMessageToProc(pcbNode *proc, proccesMsg *msg)
{
   dword last = (proc->msgQueue.last + 1) % MSG_QUEUE_SIZE;

   if( last == proc->msgQueue.first )
      proc->msgQueue.first = (proc->msgQueue.first + 1) % MSG_QUEUE_SIZE;  /**overwrite
oldest message*/

   proc->msgQueue.last = last;

   proc->msgQueue.procMsg[last].msg = msg->msg;
   proc->msgQueue.procMsg[last].msgType = msg->msgType;
   proc->msgQueue.procMsg[last].param = msg->param;


   cond_signal(&proc->msgQueue.cond_emtpyQueue);

   return true;

}
```

```
/**remove also blocked procces queue*/
void sendMessage(proccesMsg *msg)
{
   pcbNode *temp, *curr = readyProccesQueue;

   while(curr != NULL)
   {
      if(curr->pType != P_TYPE_GUI  || guiProcceses.nodes[guiProcceses.curr] == curr)
         addMessageToProc(curr, msg);

      curr = curr->next;
   }

   curr = blockedProccesQueue;
   while(curr != NULL)
   {
      temp = curr;
      curr = curr->next;

      if(temp->pType != P_TYPE_GUI  || guiProcceses.nodes[guiProcceses.curr] == temp)
         addMessageToProc(temp, msg);
   }

}

dword SccreateProcces(const char *exepath, const char *params)
{
   proc_mem_regions reg;
   byte isPitRunning = isIrqPitEnable();

   if(isPitRunning)
```

```
    PicSetIrqMask(0);
//if(kkk == 4){__asm volatile("cli;"); printk("kkk=%x", kkk); while(1);}

  pcbNode *newProc = newPcb();



  if(newProc == NULL)
    return 1;



  if(loadProcces(exepath, &reg, newProc, params) != 0)
  {
    ///??release memory and exit
    __asm volatile("cli;");
    while(1);
    return 1;
  }



//kkk++; if(kkk == 8){__asm volatile("cli;"); printk("kkk=%x", kkk); while(1);}
  initStack(&reg, newProc);
  initProcFile(newProc);
  initPcb(newProc);

  //printk("%x\n",(dword)(*((dword*)newProc->esp0 - 8)) );
  //printk("%x", (dword)(*(dword*)(newProc->esp + 48)) );
  //while(1);

  if(isPitRunning)
    PicClearIrqMask(0);

  return 0;
```

```
}

dword ScsetActiveProc(dword procHndl)
{
   guiNext(currentProcces);

   return 0;
}

dword ScsetProcProperties(dword procDescr)
{
   if(currentProcces == NULL)
      return 0;

   if( !procGuiInit(currentProcces) )
      return 0;

   currentProcces->pType = P_TYPE_GUI;
   guiNext(currentProcces);  ///?? guiNext set focus on this process
   return 1;
}

void addDriver(const char *exepath)///?? remove driver
{for(size_t i = 0;  i < 99999999; i++);
   void (*dMain)(void) = NULL;
   dword entry = 0;
kkk = 0;
   loadProcFromDisk(exepath, &entry, 1);
   dMain = entry;
   dMain();

   //__asm volatile("cli;"); printk("%s", exepath);while(1);
```

```
}


dword ScLoadDriver(const char *driverpath)
{
   addDriver(driverpath);
}
```

Αρχείο: criticalError.c

```
#include "criticalError.h"
#include "vga.h"
#include "type.h"

void criticalError(const char msg[], dword errorAdd)
{
   size_t i;
   const char CritErrMsg[] = "CRITICAL ERROR:";
   const char prom[] = "Please Restart the system";
   const size_t CritErrMsgSize = sizeof(CritErrMsg), promSize = sizeof(prom);
   const char symbolTable[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};


   setCursorPosition((word) 0, (word) 0);

   for(i = 0;  i < (size_t)(80*25) ;  i++)
      printChar(' ', 0x1F);

   setCursorPosition((word) 0, (word) 0);

   for(i = 0; i < CritErrMsgSize; i++)
      printChar(CritErrMsg[i], 0x1F);

   setCursorPosition((word) 24, (word) 0);

   for(i = 0;  i < promSize ;  i++)
      printChar(prom[i], 0x1F);

   setCursorPosition((word) 9, (word) 0);
   for(i = sizeof(dword)* 8;  i > 0;  i -= 4)
```

```
        printChar(symbolTable[(errorAdd >> (i - 4)) & 0x0F], 0x1F);

    setCursorPosition((word) 10, (word) 0);

    i = 0;
    while(msg[i] != '\0' && i < (size_t)800)
    {
        printChar(msg[i], 0x1F);
        i++;
    }

    disableCursor();

    __asm volatile("cli;");
    __asm volatile("hlt;");
}
```

Αρχείο: destroyProccess.c

```
#include "destroyProccess.h"
#include "memory.h"
#include "stdiok.h"
#include "kernelHeap.h"

extern pcbNode *currentProcces, *readyProccesQueue, *blockedProccesQueue,
*readyProccesQueueTail;

static childProccesNode* getChildNode(childProccesNode *head)
{
   childProccesNode *curr = head;

   while(curr != NULL)
   {
      if(curr->pid == currentProcces->pid)
         return curr;

      curr = curr->next;
   }

   return NULL;
}

void unblockProccess(pcbNode *pcb)  /**??*/
{
   pcbNode *curr = blockedProccesQueue, *prev = NULL;

   pcb->state = ready;

   /*find pcb of blocked proces in blockedProccesQueue and remove it*/
   while(curr != NULL && curr != pcb)
```

```
    {
        prev = curr;
        curr = curr->next;
    }

    if(curr == NULL)
        return;

    if(curr == pcb)
    {
        if(prev != NULL)
            prev->next = pcb->next;
        else
            blockedProccesQueue = blockedProccesQueue->next;
    }

    /*add pcb to readyProccesQueueTail*/
    if( readyProccesQueueTail != NULL )
    {   //printk("123\n");
        readyProccesQueueTail->next = pcb;
        pcb->next = NULL;
        readyProccesQueueTail = pcb;
    }
    else
    {
        //printk("read proc\n");
        readyProccesQueue = readyProccesQueueTail = pcb;
        pcb->next = NULL;
    }

}
```

```
static void updateParentInformation(pcbNode *pcb, int ret)
{
   childProccesNode *prntChild = NULL;

   if(pcb->parentPcb != NULL)
   {
      if(pcb->parentPcb->state == blocked)
      {
         unblockProccess(pcb->parentPcb);
      }

      if( (prntChild = getChildNode(pcb->parentPcb->childPidListHead)) == NULL)
         return;

      prntChild->hasReturn = true;
      prntChild->retVal = ret;
   }
}

static void freeChildList( childProccesNode *head)
{
   childProccesNode *temp, *curr = head;

   while(curr != NULL)
   {
      curr->pcb->parentPid = 0;
      curr->pcb->parentPcb = NULL;
      temp = curr;
      curr = curr->next;
      kfree(temp);
   }
}
```

```
static void freePcb(void)
{
   if(currentProcces != NULL && currentProcces == readyProccesQueue)
   {
      readyProccesQueue = readyProccesQueue->next;

      if(readyProccesQueue == NULL)
         readyProccesQueueTail = NULL;

      kfree(currentProcces);
      currentProcces = NULL;
   }
}

void ScdestroyProcces(int ret)///exitProcess is better
{
   if(currentProcces != NULL)
   {
      procGuiDestroy(currentProcces);
      destroyProccesCatalog();
      updateParentInformation(currentProcces, ret);
      freeChildList(currentProcces->childPidListHead);
      freePcb();

      __asm volatile("int $32;");
   }
}
```

Αρχείο: device.c

```
#include "device.h"
#include "string.h"
#include "stdiok.h"
#include "volume.h"

device devices[MAX_DEV_SUPPORTED];

int addDevice(const char* deviceId, const char* deviceName, byte devType, operations *op)
{
   size_t i = 0;

   while(i < MAX_DEV_SUPPORTED && devices[i].type != NO_DEVICE)
      i++;

   if(i >= MAX_DEV_SUPPORTED)
      return NO_FREE_SLOT_FOR_DEVICE;


   strncpy(devices[i].deviceId, deviceId, MAX_ID_STRING - 1);
   devices[i].deviceId[MAX_ID_STRING - 1] = '\0';
   strncpy(devices[i].deviceName, deviceName, MAX_NAME_STRING - 1);
   devices[i].deviceId[MAX_NAME_STRING - 1] = '\0';

   devices[i].type = devType;

   devices[i].op.getSize = op->getSize;
   devices[i].op.read = op->read;
   devices[i].op.write = op->write;
   devices[i].op.fopen = op->fopen;
   devices[i].op.fcreate = op->fcreate;
   devices[i].op.fseek = op->fseek;
```

```
    devices[i].op.fclose = op->fclose;
    devices[i].op.fexists = op->fexists;
    devices[i].op.list_dir = op->list_dir;

    if(devices[i].type == STORAGE)
    {
        strcpy(devices[i].pathId, "storage");
        inttostr(i, 10, devices[i].pathId);
        volumeSearch(devices[i].pathId);   ///??this is very temp!!!
    }

    return i;
}
```

Αρχείο: exeptions.c

```
#include "exeptions.h"
#include "stdio.h"
#include "io.h"
#include "registers.h"
#include "vga.h"
#include "criticalError.h"
#include "procScheduler.h"
#include "interupts.h"
#include "stdiok.h"

extern pcbNode *currentProcces;



static void write(dword val)
{
   const char symbolTable[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
   size_t i;
   byte *pos = (byte*)0xb8000;

   for(i = sizeof(dword)* 8;  i > 0;  i -= 4)
      {
         *pos = symbolTable[(val >> (i - 4)) & 0xF];
         pos += 2;
      }

}

void pause_system(char *mes )
{
   __asm("cli;");
```

```
   printk(mes);
   while(1);
}

INTR_HNDL void divZeroEx(struct interrupt_frame *frame)
{
   printk("cs:%x eip:%x flags:%x ss:%x esp:%x \n", (dword)  frame->cs, (dword)  frame->ip,
(dword)  frame->flags, (dword)  frame->ss, (dword) frame->sp);
   pause_system("\nError: div with 0!!\n");
   /*if(currentProcces != NULL)
   {
      //write to a file the error
      exitPorcces(255); // 255 is random
   }
   else
   {
      //panic();
   }*/
}



INTR_HNDL void debugEx(struct interrupt_frame *frame)
{
   //print info and return if defined debug
   //if ndef debug return
   pause_system("Debug exception\n");

}

/*nmi handler should be placed here*/



/*Accesed from user mode*/
```

```
INTR_HNDL void breakpointEx(struct interrupt_frame *frame)
{
   //if in user mode
   //check if current procces has install a breakpoint handler and execute it
   //else return
   pause_system("Breakpoint excetpion\n");
}

/*Accesed from user mode*/
INTR_HNDL void overflowEx(struct interrupt_frame *frame)
{
   //if current procces has install an overflow handler and call it
   //else return
   pause_system("Error: overflow\n");
}

/*Accesed from user mode*/
INTR_HNDL void boundEx(struct interrupt_frame *frame)
{
   //if current procces has install a bound exception handler and call it
   //else return
   pause_system("Error: bound exception\n");
}

INTR_HNDL void invalodOpCodeEx(struct interrupt_frame *frame)
{
   //terminate the procces and notify the user
   //if in kernel mode panic();
   //printf("Cs: %x    Ip: %x     Eflags: %x\n", (dword)frame->sp, (dword)frame->ip, (dword)frame-
>flags);
   printk("cs:%x eip:%x flags:%x ss:%x esp:%x \n", (dword)  frame->cs, (dword)  frame->ip,
(dword)  frame->flags, (dword)  frame->ss, (dword) frame->sp);
   pause_system("Error: invalid opcode\n");
```

```
}

INTR_HNDL void deviceNotAvailEx(struct interrupt_frame *frame)
{
   //terminate the procces and notify the user
   //if in kernel mode panic();
   pause_system("Error: device not available\n");
}

//error_code is always zero
INTR_HNDL void doubleFaultEx(struct interrupt_frame *frame, dword error_code)
{
   //try to terminate the procces and notify the user
   //if in kernel mode panic();
   pause_system("Error: double falt exception!\n");
}



INTR_HNDL void coprocessorOverrunEx(struct interrupt_frame *frame)
{
   //terminate the procces and notify the user
   //if in kernel mode panic();
   pause_system("Error: coprocessor overun\n");
}

/*error_code contains selector index for the segment descriptor that cause the violation
bits 0,1,2 are for EXT IDT and TI flags. See intel volume 3 chapter 5.13*/
INTR_HNDL void tssEx(struct interrupt_frame *frame, dword error_code)
{
   //terminate the procces and notify the user
   //if in kernel mode panic();
   pause_system("Error: tss exception\n");
```

```
}

/*error_code contains selector index for the segment descriptor that cause the violation
bits 0,1,2 are for EXT IDT and TI flags. See intel volume 3 chapter 5.13*/
INTR_HNDL void segmentNotPresentEx(struct interrupt_frame *frame, dword error_code)
{
   //terminate the procces and notify the user
   //if in kernel mode panic();
   pause_system("Error: segment not present exception/n");
}

/*error_code: 0 if limit voilation
         segment selector index if segment not present*/
INTR_HNDL void StackFaultEx(struct interrupt_frame *frame, dword error_code)
{
   //terminate the procces and notify the user
   //if in kernel mode panic();
   pause_system("Error: stackfault exception\n");
}

/*error_code: if fault detected while loading segment: segment selector
         0 otherwise
*/
INTR_HNDL void generalProtectionFaultEx(struct interrupt_frame *frame, dword error_code)
{
   dword eax;

   GET_EAX(eax);

   printk("cs:%x eip:%x flags:%x ss:%x esp:%x eax:%x \n", (dword)  frame->cs, (dword)  frame->ip,
(dword)  frame->flags, (dword)  frame->ss, (dword) frame->sp, eax);
   printk("error code:%x\n", error_code);
   pause_system("Error: gpf\n");
```

```
   //terminate the procces and notify the user
   //if in kernel mode panic();

}

/*error_code:?????*/
INTR_HNDL void PageFaultEx(struct interrupt_frame *frame, dword error_code)
{
   //terminate the procces and notify the user
   //if in kernel mode panic();
}

INTR_HNDL void x87FPUErrorEx(struct interrupt_frame *frame)
{
   //terminate the procces and notify the user
   //if in kernel mode panic();
   pause_system("Error: x87fpu error\n");
}

/*error_code: always 0 */
INTR_HNDL void AligmentCheckEx(struct interrupt_frame *frame, dword error_code)
{
   //terminate the procces and notify the user
   //if in kernel mode panic();
   pause_system("Error: aligment check exeption\n");
}

INTR_HNDL void MachineCheckEx(struct interrupt_frame *frame)
{
   //terminate the procces and notify the user
   //if in kernel mode panic();
   pause_system("Error: machine check exception\n");
```

```
}

INTR_HNDL void SIMDFloatPointEx(struct interrupt_frame *frame)
{
   //terminate the procces and notify the user
   //if in kernel mode panic();
   pause_system("Error: SIMD floatPiont exception\n");
}
```

Αρχείο: exportedSymbols.c

```
#include "exportedSymbols.h"
#include "memory.h"
#include "stdiok.h"
#include "interupts.h"
#include "pic.h"
#include "vga.h"
#include "createProcces.h"

symbol symbols[MAX_EPRORTED_SYMBOLS];


void initExportedSymbols(void)
{
   EXPORT_SYMBOLS_ARRAY_POINTER = (dword*) symbols;

   EXPORT_SYMBOL(printk, 0);
   EXPORT_SYMBOL(addIntGate, 1);
   EXPORT_SYMBOL(PicEoi, 2);
   EXPORT_SYMBOL(printChar, 3);
   EXPORT_SYMBOL(sendMessage, 4);
}
```

Αρχείο: fat32.c

```
#include "fat32.h"
#include "string.h"
#include "stdiok.h"
#include "vfs.h"
#include "memory.h"
#include "vFileOperations.h"


f32_state state;
file32 **openFiles;


/** FILE lstring???**/
void strtolstr(const char *str, word *lstr)
{
   while(*str != '\0')
      *lstr++ = (word) (('\0' << 8) | *str++);

   *lstr = L'\0';
}

void lstrtostr(const word *lstr, char *str)
{
   while(*lstr != L'\0')
      *str++ = (char) (*lstr++ & 0xFF);

   *str = '\0';
}

word* lstrchr(const word *str, const word lchar)
{
```

```
   do{
      if(*str == lchar  )
         return (word*) str;
   }while(*str++ != L'\0');

   return NULL;
}

size_t lstrlen(const word *str)
{
   size_t i = 0;

   while(str[i] != L'\0')
      i++;

   return i;
}

void printl(const word* lstr)
{
   size_t len = lstrlen(lstr), i;

   if(!len)
      return;

   for(i = 0;  i < len;  i++)
      printk(&lstr[i]);
}
/**FILE lstring???**/

static inline bool fat_indx_is_cached(fatCache *cache, dword indx)
{
```

```
   return (cache->startlba * (state.bytesPerSector / sizeof(dword) ) <= indx) && ( (cache->startlba
+ cache->sizeSectors) * (state.bytesPerSector / sizeof(dword) ) > indx );
}

static dword countBytesToCluster(dword bytes)
{
   return bytes / (state.bytesPerSector * state.sectorsPerCluster)
        + !!bytes % (state.bytesPerSector * state.sectorsPerCluster);
}

static bool cacheNotEqual(fatCache *fc1, fatCache *fc2)
{
   if( (fc1->sizeSectors != fc2->sizeSectors)  || (fc1->startlba != fc2->startlba) )
      return true;

   return false;
}

static void copyCache(fatCache *dest, fatCache *source )
{
   dest->sizeSectors = source->sizeSectors;
   dest->startlba = source->startlba;
   dest->dirtyBit = source->dirtyBit;
   dest->isActive = source->isActive;

   copyBytes((byte*) dest->buffer, (byte*) source->buffer, dest->sizeSectors *
state.bytesPerSector );
}

static size_t fatCacheFlush(fatCache *ftCache)
{
   size_t fatstart = state.reservedSectors * state.bytesPerSector;
```

```
   ftCache->dirtyBit = 0;
   return vfwrite(state.vPath, fatstart + ftCache->startlba * state.bytesPerSector, ftCache->sizeSectors * state.bytesPerSector, (byte *) ftCache->buffer, 0);
}

static size_t updateFatCache(fatCache *cache, dword startLba)
{
   size_t fatstart = state.reservedSectors * state.bytesPerSector, i, j;

   if(cache->dirtyBit == 1)
      fatCacheFlush(cache);

   cache->cacheHit = 0;
   cache->startlba = startLba;
   cache->dirtyBit = 0;

   //printk(" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaad\n");

//for(i = 0;  i < 40000000; i++)j = i;/**temp*/
   return vfread(state.vPath, NULL, fatstart + cache->startlba * state.bytesPerSector, cache->sizeSectors * state.bytesPerSector, (void *) cache->buffer);
}

static int fatCahceRefreshIfMiss(fatCache *cache, dword indx)
{
   size_t indxlba = fat_indx_to_lba(indx, state.bytesPerSector);   /*relative to fat table*/
   const size_t maxstartlba = state.sectorsPerFat - cache->sizeSectors;
   size_t buffersize = cache->sizeSectors * state.bytesPerSector;

   if( fat_indx_is_cached(cache, indx) )
   {
      cache->cacheHit++;
   }
```

```
    else   /**update cach*/
    {
        if(updateFatCache(cache, ((indxlba <= maxstartlba ) ? indxlba : maxstartlba) ) != buffersize)
        {
            cache->isActive = false;
            return 0;
        }
    }

    return 1;
}

static void setFatCacheValue(fatCache *cache, dword indx, dword value)
{
    if(cache->dirtyBit != 2) fatCahceRefreshIfMiss(cache, indx);

    cache->dirtyBit = 1;
    cache->buffer[indx - cache->startlba * (state.bytesPerSector / sizeof(dword) )] = value;
}

static dword getFatIndxFromCache(fatCache *cache, dword indx)
{
    if(cache->dirtyBit != 2) fatCahceRefreshIfMiss(cache, indx);

    return cache->buffer[indx - cache->startlba * (state.bytesPerSector / sizeof(dword) ) ];
}


/*returns a list of the next @count cluster indexes of the file*/
static size_t getFatList(dword first, size_t count, dword *fatList, fatCache *cache, dword
*nextFatIndx)
{
    size_t i = 0;
```

```
   if( cache == NULL || fatList == NULL )
      return 0;

   if( !cache->isActive )
      return 0;

   fatList[i] = first;

   while(fatList[i] != END_OF_FAT_LIST && (i + 1) < count)
   {
      if( (fatList[i + 1] = getFatIndxFromCache(cache, fatList[i])) == 0 )
         return 0;

      i++;
   }
//printf("123fds %x\n", fatList[i]);
   if(nextFatIndx != NULL)
      *nextFatIndx = (fatList[i] == END_OF_FAT_LIST)? END_OF_FAT_LIST :
getFatIndxFromCache(cache, fatList[i]); /**Next fat index if not at end*/
//printf("count %x\n", count);

   return i + 1;
}

static size_t getFatFromIndx(dword fHandle, size_t indx)
{
   dword next, curr = openFiles[fHandle]->firstCluster;
//printf("456 %u, %x\n\n", fHandle, indx);
   while(indx--)
   {
      if( !getFatList(curr, 0, &curr, &openFiles[fHandle]->fCache, &next) )
         return 0;
```

```
        if(next == END_OF_FAT_LIST)
            return 0;

        curr = next;
    }

    return curr;
}

static void releaseSectors(dword from, fatCache *fc, bool eoflMark)
{
    dword next, curr = getFatIndxFromCache(fc, from);
next = 1;
    if(eoflMark)
        setFatCacheValue(fc, from, END_OF_FAT_LIST);
    else
        setFatCacheValue(fc, from, FREE_FAT_ENTRY);

    while(curr != END_OF_FAT_LIST)
    {
        next = getFatIndxFromCache(fc, curr);
        setFatCacheValue(fc, curr, FREE_FAT_ENTRY);
//printf("curr: %X\n", from);
        curr = next;
    }

    fatCacheFlush(fc);
}

static dword getFreeClusterIndx(fatCache *ch)
{
```

```
   size_t s, i;
   dword clIndx;

   s = i = ch->startlba * (state.bytesPerSector / sizeof(dword) );
//ch->dirtyBit = 2;
   do{//printf("lkjh::::%x %x\n", i, getFatIndxFromCache(ch, i));
      if( (clIndx = getFatIndxFromCache(ch, i)) == FREE_FAT_ENTRY)
      {
         setFatCacheValue(ch, i, END_OF_FAT_LIST);
         ch->dirtyBit = 0;
         //printf("FAT %X\n",getFatIndxFromCache(ch, i));
         return i;
      }

      if(i == (state.sectorsPerFat * (state.bytesPerSector / (sizeof(dword) ) ) - 1 ) )
         i = 2;
      else
         i++;

   }while(i != s);

   return 0;
}

static dword reserveSectors(dword appentTo, fatCache *fc, size_t count)
{
   dword first = 0, v;
   fatCache searchCache;

   if( (searchCache.buffer = pageFrameAlloc(fc->sizeSectors / (PAGE_FRAME_SIZE /
state.bytesPerSector) ) ) == NULL )  /**better call initCache*/
   {
      searchCache.isActive = false;
```

```
        return 0;
    }

    copyCache(&searchCache, fc);

    while(count--)
    {
        v = getFreeClusterIndx(&searchCache);

        if(!first)
            first = v;
//printf("apent  %X   v %x\n\n", appentTo, v);
        if(appentTo)
            setFatCacheValue(fc, appentTo, v);

        if(cacheNotEqual(fc, &searchCache))
        {//printf("copy\n");
            fatCacheFlush(fc);
            copyCache(fc, &searchCache);
        }

        appentTo = v;
    }

    if(appentTo)
        setFatCacheValue(fc, appentTo, END_OF_FAT_LIST);

    fatCacheFlush(fc);

    return first;
}
```

```
static byte *getClusterFromCache(size_t index, clusterCache *clCache)
{
   size_t pos = index % clCache->recordsCount;

   if(clCache->clusters[pos].index == index)
      return clCache->clusters[pos].data;

   return NULL;
}

static inline bool notCachedContiniousCluster(clusterCache *clCache, size_t currCluster, size_t
prevCluster)
{
   if( (getClusterFromCache(currCluster, clCache) == NULL)  &&   (prevCluster == 0  || prevCluster
+ 1 == currCluster) )
      return true;

   return false;
}

static void flush(clusterCache *clCache)
{
   size_t i;   /**continious write??**/

   for(i = 0;  i < clCache->recordsCount; i++)
   {
      if(clCache->clusters[i].dirtyBit == 1)
      {
         //printf("writing at: %x count:%x \n", (state.reservedSectors + state.sectorsPerFat *
state.fatsCount + ((clCache->clusters[i].index - 2) * state.sectorsPerCluster )) *
state.bytesPerSector, clCache->clusterSize);
         vfwrite(state.vPath, (state.reservedSectors + state.sectorsPerFat * state.fatsCount +
((clCache->clusters[i].index - 2) * state.sectorsPerCluster )) * state.bytesPerSector, clCache-
>clusterSize, clCache->clusters[i].data, 0);
```

```
            clCache->clusters[i].dirtyBit = 0;
        }
    }
}

static void refreshCash(clusterCache *clCache)
{
    size_t i;

    for(i = 0;  i <  clCache->recordsCount;  i++)
        clCache->clusters[i].index = 0;
}

static void writeToClusterCache(clusterCache *clCache, byte *input, dword clusterId)
{
    clCache->clusters[clusterId % clCache->recordsCount].dirtyBit = 1;
    clCache->clusters[clusterId % clCache->recordsCount].index = clusterId;
    copyBytes(clCache->clusters[clusterId % clCache->recordsCount].data, input, clCache-
>clusterSize);
}

static bool updateClusterCache(clusterCache *clCache, dword *fatIndxs, size_t clustersToRead)
{
    byte *buffer;
    size_t i, br;

    if(clCache == NULL || fatIndxs == NULL)
        return false;

    if((buffer = pageFrameAlloc(clustersToRead * state.sectorsPerCluster * state.bytesPerSector /
PAGE_FRAME_SIZE + 1)) == NULL)
        return false;
```

```
   flush(clCache);
   br = vfread(state.vPath, NULL, (state.reservedSectors + state.sectorsPerFat * state.fatsCount + ((fatIndxs[0] - 2) * state.sectorsPerCluster )) * state.bytesPerSector, clustersToRead * state.sectorsPerCluster * state.bytesPerSector, (void *) buffer );


   for(i = 0;  i < clustersToRead; i++)
   {
      copyBytes(clCache->clusters[fatIndxs[i] % clCache->recordsCount].data, &buffer[i * state.sectorsPerCluster * state.bytesPerSector], state.bytesPerSector * state.sectorsPerCluster );
      clCache->clusters[fatIndxs[i] % clCache->recordsCount].index = fatIndxs[i];
      clCache->clusters[fatIndxs[i] % clCache->recordsCount].dirtyBit = 0;
   }

   pageFrameFree( (dword)buffer / PAGE_FRAME_SIZE, clustersToRead * state.sectorsPerCluster * state.bytesPerSector / PAGE_FRAME_SIZE + 1);

   return true;
}

///readClusters
#define CLUSTER_OP_READ   0x01
#define CLUSTER_OP_WRITE  0x02
static size_t OperateToClusters(byte *dstream, size_t clustersCount, dword *fatIndxs, clusterCache *clCache, size_t firstClusterOffset, size_t lastClusterLimit, size_t *bytesReaded, int operation)
{
   size_t i = 0, j;
   size_t copyFrom, copyCount;
   size_t clustersToRead = 0;
   byte *clusterBuff;

   *bytesReaded = 0;
```

```
  if(dstream == NULL || fatIndxs == NULL || clCache == NULL)
    return 0;

  while(i < clustersCount  && fatIndxs[i] != END_OF_FAT_LIST)
  {
    if( (clusterBuff = getClusterFromCache(fatIndxs[i], clCache) ) != NULL)
    {
      copyCount = state.bytesPerSector * state.sectorsPerCluster;
      if(i == 0)  /**first cluster*/
      {
        copyFrom = firstClusterOffset;
        copyCount -= copyFrom;
      }
      else
        copyFrom = 0;

      if(i == clustersCount - 1) /**Last cluster*/
        copyCount -= ( (state.bytesPerSector * state.sectorsPerCluster) - lastClusterLimit );

      if(operation == CLUSTER_OP_READ)
        *bytesReaded += copyBytes( dstream + (*bytesReaded), clusterBuff + copyFrom,
copyCount);
      else if(operation == CLUSTER_OP_WRITE)
      {
        *bytesReaded += copyBytes( clusterBuff + copyFrom, dstream + (*bytesReaded),
copyCount);
        clCache->clusters[fatIndxs[i] % clCache->recordsCount].dirtyBit = 1;
        //printf("bytes:%u\n", *bytesReaded);
      }

      i++;
    }
```

```
    else
    {
      clustersToRead = 0;

      while( notCachedContiniousCluster(clCache, fatIndxs[i + clustersToRead], (clustersToRead
? fatIndxs[i + clustersToRead - 1] : 0) ) && (i + clustersToRead < clustersCount ) &&
(clustersToRead < clCache->recordsCount ) )
        clustersToRead++;

      updateClusterCache(clCache, &fatIndxs[i], clustersToRead);

      for(j = 0;  j < clustersToRead;  j++)/**do???*/
      {
        copyCount = state.bytesPerSector * state.sectorsPerCluster;
        if(i == 0)  /**first cluster*/
        {
          copyFrom = firstClusterOffset;
          copyCount -= copyFrom;
        }
        else
          copyFrom = 0;

        if(i == clustersCount - 1) /**Last cluster*/
          copyCount -= ( (state.bytesPerSector * state.sectorsPerCluster) - lastClusterLimit );

        clusterBuff = getClusterFromCache(fatIndxs[i], clCache);

        if(operation == CLUSTER_OP_READ)
          *bytesReaded += copyBytes( dstream + (*bytesReaded), clusterBuff + copyFrom,
copyCount);
        else if(operation == CLUSTER_OP_WRITE)
        {
          *bytesReaded += copyBytes( clusterBuff + copyFrom, dstream + (*bytesReaded),
copyCount);
```

```
                clCache->clusters[fatIndxs[i] % clCache->recordsCount].dirtyBit = 1;
                //printf("bytes:%u\n", *bytesReaded);
            }
            //printf("bytes: %u\n", *bytesReaded);
            i++;
        }
    }
  }

  return i;
}

static dword readClustersFromFile(dword fromCluster, size_t countClusters, fatCache *ftCache,
clusterCache *clCache, byte *out, size_t *clustersReaded, size_t firstClusterOffset, size_t
lastClusterLimit, size_t *bytesReaded, dword *fatListRet, int operation)
{
  size_t fatListSize, br;
  dword nextFatIndx;
  dword *fatListTemp = kmalloc(countClusters * sizeof(dword) );
//printk("cl count = %x      \n", countClusters);
  if(fatListTemp == NULL)
    return 0;

  dword *fatList = (fatListRet != NULL) ? fatListRet : fatListTemp;

  if((fatListSize = getFatList(fromCluster, countClusters, fatList, ftCache, &nextFatIndx) ) <
countClusters )
    lastClusterLimit = state.sectorsPerCluster * state.bytesPerSector;

  *clustersReaded = OperateToClusters(out, fatListSize, fatList, clCache, firstClusterOffset,
lastClusterLimit, &br, operation);

  if(bytesReaded != NULL )
    *bytesReaded = br;
```

```
   kfree(fatListTemp);

   return (*clustersReaded < fatListSize)? (fatList[*clustersReaded]) : nextFatIndx;
}

static bool dirOpen(dword dirCluster, directory *dir)
{
   dir->data_buff_size = DIRECTORY_CLUSTER_BUFFER_SIZE;/**not align!!!*/
   size_t br;


   if( (dir->data = pageFrameAlloc(dir->data_buff_size)) == NULL )
      return false;

   dir->eod = false;
   dir->startCluster = dir->nextCluster = dirCluster;

   dir->nextCluster = readClustersFromFile(dir->nextCluster, ( (dir->data_buff_size *
PAGE_FRAME_SIZE) / (state.bytesPerSector * state.sectorsPerCluster) ), &state.globalFatCache,
&state.globalClusterCache, dir->data, &(dir->CountClusters), 0, state.bytesPerSector *
state.sectorsPerCluster, &br, dir->loadedFats, CLUSTER_OP_READ );


   //printk("bytes read:  %x  dir->data %x     \n", br, dir->data[4096-32]);

   dir->currEntryIndex = 0;
   dir->curr_entry = NULL; //(directoryEntry *) dir->data;

   return true;
}

static void dirClose(directory *dir)
```

```
{
   pageFrameFree( (dword) (dir->data) / PAGE_FRAME_SIZE, dir->data_buff_size);
   dir->eod = true;
}

static bool dirNext(directory *dir)
{
   if( dir->currEntryIndex * DIR_ENTRY_SIZE < (dir->CountClusters * state.sectorsPerCluster *
state.bytesPerSector) - DIR_ENTRY_SIZE)
   {
      dir->currEntryIndex++;
      return true;
   }
   else if(dir->nextCluster != END_OF_FAT_LIST)
   {
      dir->currEntryIndex = 0;
      dir->nextCluster = readClustersFromFile(dir->nextCluster, ( (dir->data_buff_size *
PAGE_FRAME_SIZE) / (state.bytesPerSector * state.sectorsPerCluster) ), &state.globalFatCache,
&state.globalClusterCache, dir->data, &(dir->CountClusters), 0, state.bytesPerSector *
state.sectorsPerCluster, NULL, dir->loadedFats, CLUSTER_OP_READ );
      return true;
   }

   dir->eod = true;
   return false;
}

static bool dirSetLfn(directory *dir)
{
   size_t longNameSize = 0;
   bool eod = false;

   while( (!eod) && dir->curr_entry[dir->currEntryIndex].data[0] != 0 &&
ENTRY_IS_LONG_NAME(dir->curr_entry[dir->currEntryIndex]) && ( dir->nextCluster ==
```

```
END_OF_FAT_LIST && (dir->currEntryIndex * DIR_ENTRY_SIZE < dir->CountClusters *
state.sectorsPerCluster * state.bytesPerSector) ) )
  {
    longNameSize += dirGetLfn( dir->curr_entry[dir->currEntryIndex], &dir-
>curr_lnf[(LDIR_GET_ORDER(dir->curr_entry[dir->currEntryIndex] ) - 1) * 13] );  /**wrong * x*/

    eod = !dirNext(dir);
  }

  dir->curr_lnf[longNameSize] = '\0';
  return !eod;
}

static bool dirSetEod(directory *dir)
{
  if( DIR_EOF(dir->curr_entry[dir->currEntryIndex])   )
  {
    dir->eod = true;
    return true;
  }

  return false;
}

static bool dirGetNextEntry(directory *dir)
{
  //size_t longNameSize = 0;
  bool invalidEntry;

  if(dir->curr_entry != NULL)
  {
    if(!dirNext(dir))
      return false;
```

```
   }
   else
      dir->curr_entry = (directoryEntry *) dir->data;
//printf("\n\n\n\n\n\nDATA:::::::%x %x %x %x \n\n", dir->curr_entry[0].data[0], dir->curr_entry[0].data[1], dir->curr_entry[0].data[2], dir->curr_entry[0].data[3] );
   do{
      invalidEntry = false;

      while( DIR_ENTRY_IS_FREE(dir->curr_entry[dir->currEntryIndex])  && dir->currEntryIndex * DIR_ENTRY_SIZE < (dir->CountClusters * state.sectorsPerCluster * state.bytesPerSector))/**bug: condition return false if dir->nextCluster == End_of_fat_list always*/
         if(!dirNext(dir))
            return false;

      if( dirSetEod(dir) )
         return  false;

      if(!dirSetLfn(dir))
         return false;

      if( dirSetEod(dir) )
         return  false;

      if(DIR_ENTRY_IS_FREE(dir->curr_entry[dir->currEntryIndex]))
         invalidEntry = true;
      else if( dir->curr_lnf[0] == '\0' )
         dirGetName(dir->curr_entry[dir->currEntryIndex] , dir->curr_lnf);

   }while( invalidEntry == true );
//printl(dir->curr_lnf);
//printk("<%x> [%x] %x |",dir->curr_entry[dir->currEntryIndex].data[0], dir->data[dir->currEntryIndex*32], dir->currEntryIndex);
//if(dir->CountClusters == 0) while(1);
   return true;
```

```
}

static int lfnCmp(const word *str1, const word *str2)
{
   size_t i = 0;

   while( str1[i] == str2[i]  )
   {
      if(str1[i] == '\0')
         return 0;

      i++;
   }

   if(str1[i] < str2[i])
      return -1;

   return 1;

}

static dword dirEntryExist(const word *lfnName, dword clusterId, dword EntryType, dword
*fsize, dword *parentDirId, size_t *clOffset, bool *isFile)
{
   directory dir;
   dword dirId = INVALID_FILE_HANDLER;
//printf("location is: %x\n",  (state.reservedSectors * state.bytesPerSector) + (state.fatsCount *
state.sectorsPerFat * state.bytesPerSector) +
//     ((state.rootCluster - 2) * state.sectorsPerCluster * state.bytesPerSector )   );

   if( !clusterId || !dirOpen(clusterId, &dir) ) ///?? dirOpen posible never close
      return INVALID_FILE_HANDLER;
   //wprintf(L"Search for: %s...\n",lfnName);
```

```
  while( dirGetNextEntry(&dir) && (dirId == INVALID_FILE_HANDLER) )
  {
    //printk("Compare (%x):", lfnCmp(lfnName, dir.curr_lnf )); printl(lfnName); printk(" with ");
printl(dir.curr_lnf); printk("\n");
    if( !lfnCmp(lfnName, dir.curr_lnf ) )/**one if statment?*/
    {
      switch(EntryType){
        case SEARCH_DIR:
          if( ENTRY_IS_DIRECTORY(dir.curr_entry[dir.currEntryIndex]) )
            dirId = DIR_GET_CLUSTER_NUM(dir.curr_entry[dir.currEntryIndex]);
          break;
        case SEARCH_FILE:
          if( ENTRY_IS_FILE(dir.curr_entry[dir.currEntryIndex]) )
            dirId = DIR_GET_CLUSTER_NUM(dir.curr_entry[dir.currEntryIndex]);
          break;
        case SEARCH_ALL:
          dirId = DIR_GET_CLUSTER_NUM(dir.curr_entry[dir.currEntryIndex]);
      } /**switch*/

      if(isFile != NULL)
        *isFile = ENTRY_IS_FILE(dir.curr_entry[dir.currEntryIndex]);

     *fsize = DIR_GET_FILE_SIZE(dir.curr_entry[dir.currEntryIndex]);

      *parentDirId = dir.loadedFats[ (dir.currEntryIndex * DIR_ENTRY_SIZE) /
(state.bytesPerSector * state.sectorsPerCluster) ];
      *clOffset = (dir.currEntryIndex * DIR_ENTRY_SIZE) % (state.bytesPerSector *
state.sectorsPerCluster);
    }
    //wprintf(L"%s,   %s\n", lfnName, dir.curr_lnf);
  }
```

```
    //*fsize = DIR_GET_FILE_SIZE(dir.curr_entry[dir.currEntryIndex]);
    //wprintf(L"size of %s is %u\n", dir.curr_lnf, dir.curr_entry[dir.currEntryIndex].data[28] );
    dirClose(&dir);
    return dirId;
}

static size_t nextPathComponent(const word* path, word *destination, bool *isLast)
{
    size_t i = 0;

    while(path[i] != '\0' && path[i] != '/' && path[i] != '\\')
    {
        destination[i] = path[i];
        i++;
    }

    *isLast = (path[i] == '\0') ? true : false;

    destination[i] = '\0';
    return  i + 1 ;
}

static bool invalidPath(const word *path)
{
    return false;
}

void setFileSize(size_t fsize, dword fileHandle)
{
    size_t bwrited;
    size_t countClToSet = countBytesToCluster(fsize);
    size_t countClCurr = countBytesToCluster(openFiles[fileHandle]->fSize);
```

```
  dword lastClIndx;


  if(countClToSet > countClCurr)
  {
    if(countClCurr)
    {
      lastClIndx = getFatFromIndx(fileHandle, countClCurr - 1);
      reserveSectors(lastClIndx, &openFiles[fileHandle]->fCache, countClToSet - countClCurr );
    }
    else
    {
      lastClIndx = reserveSectors(0, &openFiles[fileHandle]->fCache, countClToSet );
      OperateToClusters( (byte*) (&lastClIndx), 1, (dword *) &openFiles[fileHandle]-
>fdescrClIndx, &openFiles[fileHandle]->clCache,
          openFiles[fileHandle]->fdescrClOffset + 26, openFiles[fileHandle]->fdescrClOffset +
sizeof(word) + 26, &bwrited, CLUSTER_OP_WRITE);


      OperateToClusters( ((byte*) (&lastClIndx)) + 2, 1, (dword *) &openFiles[fileHandle]-
>fdescrClIndx, &openFiles[fileHandle]->clCache,
          openFiles[fileHandle]->fdescrClOffset + 20, openFiles[fileHandle]->fdescrClOffset +
sizeof(word) + 20, &bwrited, CLUSTER_OP_WRITE);


      openFiles[fileHandle]->firstCluster = openFiles[fileHandle]->currCluster = lastClIndx;
      openFiles[fileHandle]->eof = false;
      openFiles[fileHandle]->pos = 0;
    }
  }
  else if(countClToSet < countClCurr)
  {
    if(countClToSet)
    {
      lastClIndx = getFatFromIndx(fileHandle, countClToSet - 1);
      releaseSectors(lastClIndx, &openFiles[fileHandle]->fCache, true );
    }
```

```
      else
      {
         releaseSectors(openFiles[fileHandle]->firstCluster, &openFiles[fileHandle]->fCache, false );
         OperateToClusters( (byte*) &(int){0}, 1, (dword *) &openFiles[fileHandle]->fdescrClIndx,
&openFiles[fileHandle]->clCache,
                  openFiles[fileHandle]->fdescrClOffset + 20, openFiles[fileHandle]->fdescrClOffset +
sizeof(word) + 20, &bwrited, CLUSTER_OP_WRITE);

         OperateToClusters( (byte*) &(int){0}, 1, (dword *) &openFiles[fileHandle]->fdescrClIndx,
&openFiles[fileHandle]->clCache,
                  openFiles[fileHandle]->fdescrClOffset + 26, openFiles[fileHandle]->fdescrClOffset +
sizeof(word) + 26, &bwrited, CLUSTER_OP_WRITE);
      }
   }

   OperateToClusters( (byte*) (&fsize), 1, (dword *) &openFiles[fileHandle]->fdescrClIndx,
&openFiles[fileHandle]->clCache,
                  openFiles[fileHandle]->fdescrClOffset + 28, openFiles[fileHandle]->fdescrClOffset +
sizeof(size_t) + 28, &bwrited, CLUSTER_OP_WRITE);

//refresh cash??
   //flush(&state.globalClusterCache);
   refreshCash(&state.globalClusterCache);  ///??load only change size cluster
   flush(&openFiles[fileHandle]->clCache);
   openFiles[fileHandle]->fSize = fsize;
}

dword locateFile(const word *path, dword fileType, dword *fsize, dword *parentDirId, size_t
*parentDirClOffset, bool *isFile)
{
   size_t i = 0;
   dword dirId = state.rootCluster;
   word directory[MAX_PATH_COMPNENT_CHARS];
   bool isLastComponent;
```

```
   *fsize = 0;

   if( invalidPath(path) )
      return 0;

   if(path[0] == '\0')
      return dirId;

   while( ((i += nextPathComponent( &path[i], directory, &isLastComponent) ) <= lstrlen(path) + 1
) && dirId != INVALID_FILE_HANDLER)
   {

      if(isLastComponent)
         dirId = dirEntryExist(directory, dirId, fileType, fsize, parentDirId, parentDirClOffset, isFile);
      else
         dirId = dirEntryExist(directory, dirId, SEARCH_DIR, fsize, parentDirId, parentDirClOffset,
isFile);
   }
//printk("dirid: %x \n", dirId);
   return dirId;
}

static bool volumeIsFormated(const byte *volumeId)
{
   if( (* (word *) (volumeId + OFFSET_SIGNATURE) ) != SIGNATURE )
      return false;

   return true;
}

static bool initFatCache(fatCache *ftCache, dword startLba)
{
```

```
   ftCache->sizeSectors = GLOBAL_CACHE_SEC_SIZE;
   ftCache->startlba = startLba;

   if( (ftCache->buffer = pageFrameAlloc(ftCache->sizeSectors / (PAGE_FRAME_SIZE /
state.bytesPerSector) ) ) == NULL ) /**bug + 1 pageframealloc**/
      return ftCache->isActive = false;

   ftCache->dirtyBit = 0;
   return ftCache->isActive = !!updateFatCache(ftCache, startLba);
}

static bool initClusterCache(clusterCache *clCache)
{
   size_t i = 0;

   clCache->clusterSize = state.bytesPerSector * state.sectorsPerCluster;
   clCache->recordsCount = GLOBAL_CLUSTER_MAX_RECORDS;

   for( i = 0;  i < clCache->recordsCount;  i++ )
   {
      clCache->clusters[i].index = 0;
      clCache->clusters[i].dirtyBit = 0;
      clCache->clusters[i].data = pageFrameAlloc( (state.sectorsPerCluster * state.bytesPerSector)
/ PAGE_FRAME_SIZE );
   }
   return true;
}

bool init(const char * volume)
{
   byte *volumeId;
   size_t i;
```

```
   /**openFiles init*/
   openFiles = pageFrameAlloc( (sizeof(file32*) * FS_MAX_OPEN_FILES ) / PAGE_FRAME_SIZE + 1
);
   for(i = 0;  i < FS_MAX_OPEN_FILES; i++)
      openFiles[i] = NULL;

   if( ( volumeId = pageFrameAlloc(1) ) == NULL )
   {
      return false;
   }

   vfread(volume, NULL, 0, 512, (void *) volumeId);
   /*check if vfread fail and return*/

   for(i = 0;  i < 100000; i++);/*wait ide*/

   state.bytesPerSector = (* (word *) (volumeId + OFFSET_BYTES_PER_SECTOR) );
   state.sectorsPerCluster = * (volumeId + OFFSET_SECTORS_PER_CLUSTER) ;
   state.fatsCount = * (volumeId + OFFSET_FATS_COUNT) ;
   state.reservedSectors = (* (word *) (volumeId + OFFSET_RESERVED_SECTORS ) );
   state.sectorsPerFat = (* (dword *) (volumeId + OFFSET_SECTORS_PER_FAT) );
   state.rootCluster = (* (dword *) (volumeId + OFFSET_ROOT_DIR_CLUSTER) );
   state.vPath[0] = '\0';
   strcpy(state.vPath, volume);
   state.isformated = volumeIsFormated(volumeId);

   pageFrameFree((dword)volumeId / PAGE_FRAME_SIZE, 1);

   initFatCache(&state.globalFatCache, 0);
   initClusterCache(&state.globalClusterCache);

   return true;
}
```

```
dword file_exists(const char *fpath, dword *attr, size_t *fsize)
{
   const word lfpath[128];
   size_t parDirClOffset;
   dword parDirClIndx, cfsize, res;
   bool isFile;

   strtolstr(fpath, lfpath);

   res = locateFile(lfpath, SEARCH_ALL, &cfsize, &parDirClIndx, &parDirClOffset, &isFile);

   if( res == INVALID_FILE_HANDLER  /*|| res == 0*/ )
      return 0;

   if(attr != NULL)
      *attr = (dword) isFile;

   if(fsize != NULL)
      *fsize = cfsize;

   return 1;
}

dword list_dir(const char* fpath, const dword fromFile, size_t countFiles,  fileAttr *fileAtt)
{
   const word lfpath[128];
   size_t parDirClOffset, i = 0, tot = 0;
   dword parDirClIndx, clusterId, fsize = 0;
   directory dir;

   strtolstr(fpath, lfpath);
```

```
   clusterId = locateFile(lfpath, SEARCH_DIR, &fsize, &parDirClIndx, &parDirClOffset, NULL);
   if( clusterId == INVALID_FILE_HANDLER || !clusterId)
      return INVALID_FILE_HANDLER;

   if(!dirOpen(clusterId, &dir) ) ///?? dirOpen posible never close
      return INVALID_FILE_HANDLER;

   while( dirGetNextEntry(&dir) && (i < fromFile) )
      i++;



   while( dirGetNextEntry(&dir)  && (i < fromFile + countFiles) )
   {
      lstrtostr(dir.curr_lnf, fileAtt[tot++].fName );
      i++;
   }



   dirClose(&dir);
   return tot;
}

dword file_open(const char* fpath)
{
   const word lfpath[128];
   size_t parDirClOffset, i = 0, j = 0;
   dword parDirClIndx, clusterId, fsize = 0;

   strtolstr(fpath, lfpath);
//printf("------pass5----------\n");
```

```
  if( (clusterId = locateFile(lfpath, SEARCH_FILE, &fsize, &parDirClIndx, &parDirClOffset, NULL))
== INVALID_FILE_HANDLER )
     return INVALID_FILE_HANDLER;

  while(openFiles[i] != NULL  && i < FS_MAX_OPEN_FILES)
     i++;

  if(i == FS_MAX_OPEN_FILES)
     return INVALID_FILE_HANDLER;

  if( NULL == (openFiles[i] = kmalloc(sizeof(file32) ) ) )
     return INVALID_FILE_HANDLER;

  openFiles[i]->firstCluster = openFiles[i]->currCluster = clusterId;
  openFiles[i]->currClusterOffset = 0;
  openFiles[i]->fSize = fsize;
  openFiles[i]->pos = 0;
  openFiles[i]->fdescrClOffset = parDirClOffset;
  openFiles[i]->fdescrClIndx = parDirClIndx;

  openFiles[i]->eof = (!fsize) ? true : false;

  initFatCache(&openFiles[i]->fCache, 0);
  initClusterCache(&openFiles[i]->clCache);


  /*************************************
  printf("------File Open----------\n");
  dword list[14];
  size_t max = getFatList(openFiles[i]->firstCluster, 14, list, &openFiles[i]->fCache, NULL);
  //for(size_t k = 0;  k < max;  k++)
     //printf("%X\n", list[k]);
```

```
   ************************************/


   return i;
}

int file_seek(dword fileHandle, long64 offset, size_t origin)
{
   dword clustersOffset, nextCluster;
   dword *fatList = NULL;
   dword offset32;
   long64 absPos;


   if(origin == SEEK_SET)
      absPos = offset;
   else if(origin == SEEK_CUR)
      absPos = openFiles[fileHandle]->pos + offset;
   else if(origin == SEEK_END)
      absPos = openFiles[fileHandle]->fSize + offset;
   else
      return -1;

   if( absPos < 0 )
      absPos = 0;

   offset32 = absPos;

   if(offset32 > openFiles[fileHandle]->fSize)
   {
      openFiles[fileHandle]->eof = true;
      offset32 = openFiles[fileHandle]->fSize;
```

```
   }
   else
      openFiles[fileHandle]->eof = false;

   if( (clustersOffset = offset32 / (state.sectorsPerCluster * state.bytesPerSector)) != 0 )
   {
      if(  (fatList = kmalloc(clustersOffset * sizeof(dword) ))  == NULL  )
         return -1;

      if(getFatList(openFiles[fileHandle]->firstCluster, clustersOffset, fatList,
&openFiles[fileHandle]->fCache, &nextCluster ) != clustersOffset)
      {
         kfree(fatList);
         return -1;
      }

      openFiles[fileHandle]->currCluster = nextCluster;
   }
   else
      openFiles[fileHandle]->currCluster = openFiles[fileHandle]->firstCluster;

   openFiles[fileHandle]->currClusterOffset = offset32 % (state.sectorsPerCluster *
state.bytesPerSector);


   kfree(fatList);
   openFiles[fileHandle]->pos = offset32;
   return openFiles[fileHandle]->pos;
}

static void splitLastPathComponent(const word* fpath, word* folder, word* fileName)
{
   size_t i = 0, j = 0, max = 0;
```

```
   fileName[0] = folder[0] = '\0';

   while(fpath[i] != '\0')
         max = ++i;

   while(fpath[i] != L'/' && fpath[i] != L'\\' && i > 0)
      i--;

   while(j < i)
   {
      folder[j] = fpath[j];
      j++;
   }

   folder[j] = '\0';
   i = 0;

   if(fpath[j] == L'/' || fpath[j] == '\\')
      j++;

   while(j <= max)
      fileName[i++] = fpath[j++];

}

static size_t expandDir(directory *dir, size_t count)
{
   size_t BytesRemain = ((state.bytesPerSector * state.sectorsPerCluster * dir->CountClusters) - (
(dir->currEntryIndex + 1)) * DIR_ENTRY_SIZE), clReaded;
   size_t firstClOffset = ( (dir->currEntryIndex + count ) * DIR_ENTRY_SIZE) %
(state.bytesPerSector * state.sectorsPerCluster );
   dword cluster;
```

```
   if(BytesRemain < count  * DIR_ENTRY_SIZE)
      cluster = reserveSectors(dir->loadedFats[dir->CountClusters], &state.globalFatCache, 1  );
   else
      cluster = dir->loadedFats[dir->CountClusters];


   readClustersFromFile(cluster, 1, &state.globalFatCache, &state.globalClusterCache,
                  &(byte){0}, &clReaded, firstClOffset, firstClOffset + 1, NULL, NULL,
CLUSTER_OP_WRITE);

   flush(&state.globalClusterCache);
   return count;
}

static void getFreeEntries(dword dirId, size_t entriesNum, dword *clusterId, size_t *clusterOffset
)
{
   directory dir;
   size_t freeEntriesCount = 0;

   *clusterId = 0;

   if(!dirOpen(dirId, &dir) )
      return;

   dir.curr_entry = (directoryEntry *) dir.data;

   do{

      while( !dirSetEod(&dir) && !DIR_ENTRY_IS_FREE(dir.curr_entry[dir.currEntryIndex]) &&
(dir.nextCluster != END_OF_FAT_LIST || (dir.nextCluster == END_OF_FAT_LIST &&
(dir.currEntryIndex * DIR_ENTRY_SIZE < dir.CountClusters * state.sectorsPerCluster *
state.bytesPerSector) ) ) )
```

```
        dirNext(&dir);   /**?? Ignoring return value (if failed because of no more entries)**/


    *clusterId = dir.loadedFats[ (dir.currEntryIndex * DIR_ENTRY_SIZE) / (state.bytesPerSector *
state.sectorsPerCluster) ];
    *clusterOffset = (dir.currEntryIndex * DIR_ENTRY_SIZE) % (state.bytesPerSector *
state.sectorsPerCluster);


    while( (!dirSetEod(&dir) && DIR_ENTRY_IS_FREE(dir.curr_entry[dir.currEntryIndex]) &&
(dir.nextCluster != END_OF_FAT_LIST || (dir.nextCluster == END_OF_FAT_LIST &&
(dir.currEntryIndex * DIR_ENTRY_SIZE < dir.CountClusters * state.sectorsPerCluster *
state.bytesPerSector) ) ))  && (freeEntriesCount < entriesNum) )
    {
        freeEntriesCount++;
        dirNext(&dir);
    }


    if(dirSetEod(&dir) && (freeEntriesCount < entriesNum) )
        freeEntriesCount += expandDir(&dir, entriesNum - freeEntriesCount);
    else if(freeEntriesCount < entriesNum)
    {
        freeEntriesCount = 0;
        dirNext(&dir);
        *clusterId = 0;
    }


  }while(*clusterId == 0);



  dirClose(&dir);


}


bool UCStoAscii(const word *fileName, char *sfn)
{
```

```
    size_t i = 0;

    while(fileName[i] != L'\0')
    {
        if(fileName[i] > 127 )
            return false;

        sfn[i] = (char) fileName[i] & 0x7F;
        i++;
    }

    return true;
}

bool convertToShortFileName(const word *fileName, char *sfn, byte *lo_flag)
{
    size_t i;
    word *ext = lstrchr(fileName, L'.');

    for(i = 0;  i < DIR_NAME_MAX_CHARS; i++)
        sfn[i] = 0x20;
    sfn[i] = '\0';

    if(ext != NULL)
    {
        *ext++ = L'\0';
        if(lstrlen(ext) > DIR_NAME_EXTENSION_MAX_CHARS )
            return false;

        if(lstrchr(ext, L'.') != NULL)
            return false;
```

```
    if( !UCStoAscii(ext, sfn + DIR_NAME_MAIN_PART_MAX_CHARS) )
        return false;
}

if(lstrlen(fileName) > DIR_NAME_MAIN_PART_MAX_CHARS )
    return false;

if( !UCStoAscii(fileName, sfn) )
    return false;

*lo_flag = (1 << 4) & (1 << 3);
for(i = 0;  i < DIR_NAME_MAIN_PART_MAX_CHARS;  i++)
    if( IS_UP_CA_CHAR(sfn[i]) )
        *lo_flag &= ~(1 << 3);

for(i = DIR_NAME_MAIN_PART_MAX_CHARS;  i < DIR_NAME_MAX_CHARS;  i++)
    if( IS_UP_CA_CHAR(sfn[i]) )
        *lo_flag &= ~(1 << 4);

for(i = 0;  i < DIR_NAME_MAX_CHARS;  i++)
{
    if( !IS_VALID_SFN_CHAR(sfn[i]) && sfn[i] != 0x20)
        return false;

    if( IS_LO_CA_CHAR(sfn[i]) )
        sfn[i] = TO_UP_CA_ASCII(sfn[i]);
}

if(ext != NULL)
    *(ext - 1) = '.';

return true;
```

```
}

dword file_create(const char* fpath)
{
   dword sz = 0, parent = 0, dirId, clusterId;
   const word lfpath[128];
   size_t parentoffset = 0, clReaded;
   size_t entriesNum = 1, clusterOffset;
   word fileName[MAX_PATH_COMPNENT_CHARS];
   word createTo[MAX_PATH_COMPNENT_CHARS];
   char sfn[DIR_NAME_MAX_CHARS + 1];
   byte dirDescr[DIR_ENTRY_SIZE];


   strtolstr(fpath, lfpath);
   splitLastPathComponent(lfpath, createTo, fileName);

   if(fileName[0] == '\0')
      return 0;

   if( !convertToShortFileName(fileName, sfn, &dirDescr[12]) )
      return 0;

   strcpy((char*) dirDescr, sfn);
   dirDescr[11] = 0x20;

   dirId = locateFile(createTo, SEARCH_DIR, &sz, &parent, &parentoffset, NULL);

   if(dirId == INVALID_FILE_HANDLER)
      return 0;

   if( locateFile(lfpath, SEARCH_ALL, &sz, &parent, &parentoffset, NULL) !=
INVALID_FILE_HANDLER)
```

```
      return 0;

//printk("INSIDE\n"); printl(lfpath);   __asm volatile("cli;"); while(1);

   getFreeEntries(dirId, entriesNum, &clusterId, &clusterOffset );

   readClustersFromFile(clusterId, 1, &state.globalFatCache, &state.globalClusterCache,
                  dirDescr, &clReaded, clusterOffset, clusterOffset + DIR_ENTRY_SIZE, NULL, NULL,
CLUSTER_OP_WRITE);

   flush(&state.globalClusterCache);

   return 1;
}

static void calcLimit(size_t countBytes, size_t *clustersCount, size_t *copyLimit, file32 *fHandle )
{
   size_t bytesRemain, clusterSize = state.bytesPerSector * state.sectorsPerCluster;

   *clustersCount = 1;

   if(countBytes + fHandle->currClusterOffset <= clusterSize )
      *copyLimit = countBytes + fHandle->currClusterOffset;
   else
   {
      bytesRemain = countBytes - ( (clusterSize) - fHandle->currClusterOffset );
      *clustersCount += bytesRemain / (clusterSize);
      if( (*copyLimit = bytesRemain % (clusterSize) ) )
         ++*clustersCount;
      else
         *copyLimit = clusterSize;
   }
}
```

```
size_t file_read(dword fileHandle, size_t countBytes, byte *out)
{
   size_t bytesRemain, copyLimit, clustersCount = 1;
   size_t clustersReaded, bytesReaded, clusterSize = state.bytesPerSector *
state.sectorsPerCluster;



   if( openFiles[fileHandle]->pos + countBytes >= openFiles[fileHandle]->fSize)
      countBytes = openFiles[fileHandle]->fSize - openFiles[fileHandle]->pos;


   if(openFiles[fileHandle]->eof)
      return 0;
//byte *p = malloc(4096); for(int i =0; i < 4096;  i++)p[i] = '&';
writeToClusterCache(&(openFiles[fileHandle]->clCache), p, 600 ); free(p);
   if(countBytes + openFiles[fileHandle]->currClusterOffset <= clusterSize )
      copyLimit = countBytes + openFiles[fileHandle]->currClusterOffset;
   else
   {
      bytesRemain = countBytes - ( (clusterSize) - openFiles[fileHandle]->currClusterOffset );
      clustersCount += bytesRemain / (clusterSize);
      if( (copyLimit = bytesRemain % (clusterSize) ) )
         clustersCount++;
      else
         copyLimit = clusterSize;
   }


   /**calc clusters count*/
   //clustersCount =
   //printf("File Read...\n");
   readClustersFromFile(openFiles[fileHandle]->currCluster, clustersCount,
&(openFiles[fileHandle]->fCache), &(openFiles[fileHandle]->clCache), out, &clustersReaded,
openFiles[fileHandle]->currClusterOffset, copyLimit, &bytesReaded, NULL, CLUSTER_OP_READ  );
   //printf("file read: %u,   %u\n", bytesReaded, openFiles[fileHandle]->pos);
```

```
   file_seek(fileHandle, (long64) (openFiles[fileHandle]->pos + bytesReaded), SEEK_SET );



   return bytesReaded;
}

size_t file_write(dword fileHandle, size_t countBytes, byte *input, int flags)
{
   size_t clusterCount, copyLimit, clRead, br = 0;

   if(flags == 1)
   {
      file_seek(fileHandle, (long64) 0, SEEK_SET );
      setFileSize(0, fileHandle);
      //printf("\n\n File size is 0\n");
      //while(1);
      //return 0;
   }

   if( openFiles[fileHandle]->pos + countBytes >= openFiles[fileHandle]->fSize)
      setFileSize(openFiles[fileHandle]->pos + countBytes, fileHandle);



   calcLimit(countBytes, &clusterCount, &copyLimit, openFiles[fileHandle] );



   readClustersFromFile(openFiles[fileHandle]->currCluster, clusterCount,
&openFiles[fileHandle]->fCache, &openFiles[fileHandle]->clCache, input, &clRead,
openFiles[fileHandle]->currClusterOffset,
                  copyLimit, &br, NULL, CLUSTER_OP_WRITE);
```

```
    file_seek(fileHandle, (long64) (openFiles[fileHandle]->pos + br), SEEK_SET );
    flush(&openFiles[fileHandle]->clCache);

    return br;
}

void file_close(dword fileHandle)
{
    kfree(openFiles[fileHandle]);

    openFiles[fileHandle] = NULL;
}

size_t installFat32(const char *vpath)
{
    operations op;

    //op.getSize = getSize;
    op.read = file_read;
    op.write = file_write; //writeSectors;
    op.fopen = file_open;
    op.fseek = file_seek;
    op.fcreate = file_create;
    op.fclose = file_close;
    op.fexists = file_exists;
    op.list_dir = list_dir;
    //op.setSize = setFileSize;

    init(vpath);
    return (size_t) addDevice("FAT_32", "FAT_32", 0x5, &op);

}
```

Αρχείο: fat32dir.c

```
#include "fat32dir.h"
#include <string.h>
#include <stdio.h>
```

```
static bool isInvalidNameChar(char ch)
{
   size_t i;
   const char invalid_chars[] = {DIR_INVALID_NAME_CHARS};

   for(i = 0;  i < sizeof(invalid_chars);  i++ )
      if(ch == invalid_chars[i])
         return true;

   return false;
}

static bool dirNameIsValid(directoryEntry entry)
{
   size_t i;

   if(entry.data[0] == DIR_ENTRY_SPACE_CHAR  || isInvalidNameChar(entry.data[0]) ||
(entry.data[0] < DIR_ENTRY_SPACE_CHAR && entry.data[0] != DIR_KANJI_LEAD_BYTE) )
      return false;

   for(i = 1;  i < DIR_NAME_MAX_CHARS;  i++)
      if( isInvalidNameChar(entry.data[i]) || entry.data[i] < DIR_ENTRY_SPACE_CHAR )
         return false;

   return true;
}

word* dirGetName(directoryEntry entry, word* destination)
{
   size_t j = DIR_NAME_MAIN_PART_MAX_CHARS;
   size_t i = DIR_NAME_MAX_CHARS;
   size_t k;
```

```
   if( destination == NULL  || !dirNameIsValid(entry) )
      return NULL;

   while(entry.data[j - 1] == DIR_ENTRY_SPACE_CHAR  && j > 0)
      j--;

   for(k = 0;  k < j;  k++)
     destination[k] = entry.data[k];

   while(entry.data[i - 1] == DIR_ENTRY_SPACE_CHAR  && i >
DIR_NAME_MAIN_PART_MAX_CHARS)
      i--;

   if(i > DIR_NAME_MAIN_PART_MAX_CHARS)
   {
      destination[k++] = L'.';
      for( j = DIR_NAME_MAIN_PART_MAX_CHARS; j < i;  j++ )
         destination[k++] = entry.data[j];
   }

   destination[k] = '\0';

   return destination;
}

static bool readLDirNameEntryRegions(directoryEntry entry, word* destination, size_t
startRegion, size_t endRegion, size_t *cursor)
{
   size_t i;
   word lnf_char;

   for(i = startRegion;  i <= endRegion;  i +=2)
```

```
   {
      lnf_char = (word) ((word) entry.data[i + 1] << 8) | (word) entry.data[i];
//wprintf( L"LNFnnnnn::     %x\n", 123);
      if( lnf_char == 0x00)
         return true;

      destination[*cursor] = lnf_char;
      (*cursor)++;
   }

   return false;
}

size_t dirGetLfn(directoryEntry entry, word* destination)
{
   size_t entryCursor = 0;
   bool endOfLfnEntry;

   endOfLfnEntry = readLDirNameEntryRegions(entry, destination, LDIR_NAME1,
LDIR_NAME1_LAST, &entryCursor);

   if( !endOfLfnEntry )
      endOfLfnEntry = readLDirNameEntryRegions(entry, destination, LDIR_NAME2,
LDIR_NAME2_LAST, &entryCursor);

   if( !endOfLfnEntry )
      endOfLfnEntry = readLDirNameEntryRegions(entry, destination, LDIR_NAME3,
LDIR_NAME3_LAST, &entryCursor);

   return entryCursor;
}
```

Αρχείο: file.c

```
#include "file.h"
#include "string.h"
#include "stdiok.h"
```

```
extern pcbNode *currentProcces;

dword initProcFile(pcbNode *newProc)
{
   size_t i;

   if(newProc == NULL)
      return 0;

   for(i = 0;  i < MAX_OPEN_FILES;  i++)
   {
      newProc->openfiles[i].fpath[0] = '\0';
      newProc->openfiles[i].fileHndlr = 0;
      newProc->openfiles[i].pos = 0;
      newProc->openfiles[i].fsIndx = INVLD_FILE_INDX;
   }

   return 1;
}

dword addFiletoPcb(const char* fpath, size_t fsize, dword fsIndx)
{
   size_t i = 0;

   if(currentProcces == NULL)
      return 0;

   while(i < MAX_OPEN_FILES)
   {
      if(currentProcces->openfiles[i].fileHndlr == 0)
      {
         if(strlen(fpath) >= PATH_MAX_LENGTH)
```

```
            return 0;

        currentProcces->openfiles[i].fileHndlr = i + 1;
        strcpy(currentProcces->openfiles[i].fpath, fpath);
        currentProcces->openfiles[i].pos = 0;
        currentProcces->openfiles[i].fsize = fsize;
        currentProcces->openfiles[i].fsIndx = fsIndx;
        return i + 1;
      }
      i++;
   }
   return 0;
}

dword updateFileCursor(const char *path, size_t pos)
{
   size_t i;

   if(currentProcces == NULL)
      return 0;

   for(i = 0;  i < MAX_OPEN_FILES;  i++)
   {
      if(!strcmp(path, currentProcces->openfiles[i].fpath))
      {

         if(currentProcces->openfiles[i].fsize <= pos)
            return 0;

         currentProcces->openfiles[i].pos = pos;

         return 1;
```

```
      }
   }
   return 0;
}

const char *getPathFromHandle(dword fileHndlr)
{
   size_t i;

   if(currentProcces == NULL)
      return NULL;

   for(i = 0;  i < MAX_OPEN_FILES;  i++)
   {
      if(currentProcces->openfiles[i].fileHndlr == fileHndlr)
         return currentProcces->openfiles[i].fpath;
   }
   return NULL;
}

size_t getPosFromHandle(dword fileHndlr)
{
   size_t i;

   if(currentProcces == NULL)
      return INVLD_FILE_CUR_POS;

   for(i = 0;  i < MAX_OPEN_FILES;  i++)
   {
      if(currentProcces->openfiles[i].fileHndlr == fileHndlr)
         return currentProcces->openfiles[i].pos;
```

```
    }
    return INVLD_FILE_CUR_POS;
}

dword getFsIndFromPath(const char *path)
{
    size_t i;

    if(currentProcces == NULL)
        return PATH_NOT_FOUND;

    for(i = 0;  i < MAX_OPEN_FILES;  i++)
    {
        if( strcmp(currentProcces->openfiles[i].fpath, path) == 0 )
            return currentProcces->openfiles[i].fsIndx;
    }

    return PATH_NOT_FOUND;
}
```

Αρχείο: fileSystem.c

```
#include "fileSystem.h"

/**merge with devide.c ??*/
```

```
fileSystem fileSystems[MAX_FS_SUPPORTED];

int addFileSystem(const char* fsId, const char* fsName, byte fsType, operations *op)
{
   size_t i = 0;

   while(i < MAX_FS_SUPPORTED && fileSystems[i].type != 0x00)
      i++;

   if(i >= MAX_FS_SUPPORTED)
      return -1; //NO_FREE_SLOT_FOR_DEVICE;

   strncpy(fileSystems[i].fsId, fsId, MAX_FILE_SYSTEM_ID_STRING - 1);
   fileSystems[i].fsId[MAX_FILE_SYSTEM_ID_STRING - 1] = '\0';
   strncpy(fileSystems[i].fsName, fsName, MAX_FILE_SYSTEM_NAME - 1);
   fileSystems[i].fsId[MAX_FILE_SYSTEM_NAME - 1] = '\0';

   fileSystems[i].type = fsType;
   fileSystems[i].op.getSize = op->getSize;
   fileSystems[i].op.read = op->read;
   fileSystems[i].op.write = op->write;
   fileSystems[i].op.fopen = op->fopen;



   return 0;
}
```

Αρχείο: gui.c

```
#include "gui.h"
#include "vga.h"
```

```
guiProccesesBuf guiProcceses;
extern pcbNode *currentProcces, *readyProccesQueue, *blockedProccesQueue; /**??include file*/

void initVideo(void)
{
   size_t i;

   for(i = 0;  i < MAX_GUI_PROCCESES;  i++)
      guiProcceses.nodes[i] = NULL;

   guiProcceses.curr = guiProcceses.last = NO_SELECTED_PROCCESS;

}

static bool initVideoBuffer(pcbNode *proc)
{
   proc->videoBuffer = kmalloc(80 * 25 * 2);

   if(proc->videoBuffer == NULL)
      return false;

   return true;
}

static void freeVideoBuffer(pcbNode *proc)
{
   kfree(proc->videoBuffer);
}
extern int kkk;
static bool guiAddProc(pcbNode *currentProcces)
{
   if(guiProcceses.last == NO_SELECTED_PROCCESS)
```

```
      guiProcceses.last = 0;
   else
      guiProcceses.last++;

   if(guiProcceses.last == MAX_GUI_PROCCESES)
   {
      guiProcceses.last--;
      return false;
   }
//kkk++; if(kkk == 5){printk("last proc last %x    proc   %x  \n curent proc %x  \n",
guiProcceses.last, currentProcces, guiProcceses.curr);  __asm volatile("cli;"); while(1);}
   guiProcceses.nodes[guiProcceses.last] = currentProcces;
   return true;
}

void guiFlush(pcbNode *guiP)
{
   if(guiProcceses.nodes[guiProcceses.curr] == guiP)
   {
      //printk("Draw test10!!\n" );
      vga_write(guiP->videoBuffer);
   }
}

bool guiNext(pcbNode *currentProcces)
{
   size_t i;
//kkk++; if(kkk == 17){printk("last proc last %x    proc   %x  \n curent proc %x  \n",
guiProcceses.last, currentProcces, guiProcceses.curr);  __asm volatile("cli;"); while(1);}
   if(guiProcceses.last == NO_SELECTED_PROCCESS)
      return false;
//printk("gui next...  %x last %x", guiProcceses.curr, guiProcceses.last);
   if(guiProcceses.curr == NO_SELECTED_PROCCESS || guiProcceses.curr >= guiProcceses.last)
```

```
        guiProcceses.curr = 0;
    else if(guiProcceses.curr < guiProcceses.last )
        guiProcceses.curr++;
    else
        return false;

    guiFlush(guiProcceses.nodes[guiProcceses.curr]);
    setCursorPosition(0, 0);

    return true;

}

bool procGuiInit(pcbNode *currentProcces)
{
    if(!guiAddProc(currentProcces))
        return false;

    if(!initVideoBuffer(currentProcces))
        return false;


    return true;
}

static bool guiRemoveProc(pcbNode *currentProcces)
{
    size_t i = 0;

    if(guiProcceses.last == NO_SELECTED_PROCCESS)
        return false;
```

```
   while(i <= guiProcceses.last  &&  guiProcceses.nodes[i] != currentProcces)
      i++;

   i++;
   while(i <= guiProcceses.last)
      guiProcceses.nodes[i-1] = guiProcceses.nodes[i++];



   if(guiProcceses.last == 0)
      guiProcceses.last = NO_SELECTED_PROCCESS;
   else
      guiProcceses.last--;
//kkk++; if(kkk == 4){printk("last proc last %x    proc   %x  \n curent proc %x  \n",
guiProcceses.last, currentProcces, guiProcceses.curr);  __asm volatile("cli;"); while(1);}
   guiNext(currentProcces);

   return true;
}

void procGuiDestroy(pcbNode *currentProcces)
{
   freeVideoBuffer(currentProcces);
   guiRemoveProc(currentProcces);
}

void ScDraw(byte *buffer, size_t offset, size_t count, dword position)
{
   size_t i;

   offset *= 2;

   for(i = 0;  i < count;  i++)
   {
```

```
        currentProcces->videoBuffer[offset++] = buffer[i];
        currentProcces->videoBuffer[offset++] = 0x0F;
    }

    guiFlush(currentProcces);
    setCursorPosition((word) (position >> 16), (word) (position & 0xFFFF) );
}
```

Αρχείο: ide.c

```
#include "ide.h"
#include "segments.h"
#include "interupts.h"
#include "pic.h"
```

```
#include "stdiok.h"
#include "ioblock.h"
#include "device.h"
#include "memory.h"
#include "registers.h"

diskInfo dskinf;
bool wait_interupt;

struct commandState{
   byte     lastCommand; /**Last command  executed*/
   size_t   bytesReadWriteSucced; /**Number of bytes succesfully readed or writed */
   byte     *ata_buff; /**buffef to write or read*/
   byte     *tempSector;
   size_t   buffer_index;
   size_t   firstByte, lastByte; /**start read/write from firstByte at first sector stop read/write at lastByte at last sector*/
   size_t   firstSector, lastSector; /**start read/write from sector firstSector stop read/write to sector lastSector*/
   size_t   curr_sector;
   size_t   sectorsCount;      /**Number of sectors to read or write per command(outside readsectors() is read only)*/
   bool     identifyDriveComlete; /**dskinf struct contains valid data*/
} cmndSt;

#define INTR_HNDL __attribute__((interrupt, no_caller_saved_registers))
INTR_HNDL void ide_handler(struct interrupt_frame*);

/*Sends a command to the ata controler*/
void sendComand(byte command)
{
   cmndSt.lastCommand = command;
   /*check if drive is ready(bsy bit)*/
```

```
   writeCommandReg(command);
}

void identify_dr(void)
{

   readStatusReg(); /*clear any pending interrupt*/
   /*wait until driver not busy function need here*/
   cmndSt.identifyDriveComlete = false;
   cmndSt.sectorsCount = 1;
   sendComand(IDENTIFY_DRIVE);

   while(cmndSt.identifyDriveComlete == false);

}

/*how to find base addersses and irq number??*/
void initDisk(void)
{
   writeDeviceControlReg(8);/*enable ata interrupts*/
   addIntGate(PRIMARY_ATA_IRQ, CS_PL0, (dword) ide_handler,  INT_GATE | DPL0);
   identify_dr();
}

void enableLba(void)
{
   byte temp = readDriveHeadReg();

   writeDriveHeadReg( (temp |= (1 << 6)) );
}
```

```
/*this function called only from interrupt handler
to initialize dskInf struct when executing the identify_drive command */
void setDskInf(void)
{
   size_t i;
   word temp;

   readDataReg();  /*word 0*/
   dskinf.cyl_cnt = readDataReg();
   readDataReg(); /*word 2 ata reserved*/
   dskinf.head_cnt = readDataReg();
   readDataReg(); /*unformatted bytes per track*/
   readDataReg(); /*unformatted bytes per sector*/
   dskinf.spt_cnt = readDataReg();
   for(i = 7;  i <= 9; i++) readDataReg();/*words 7 to 9 ata reserved*/
   for(i = 0; i < 10;  i++)
      dskinf.serial_num[i] = readDataReg();
   readDataReg(); /*buffer type (word 20)*/
   dskinf.buffer_sz = readDataReg();
   readDataReg();
   readDataReg();/*word 23*/
   for(i = 0; i < 40;  i += 2){
      temp = readDataReg();
      dskinf.model_num[i] = (char) (temp >> 8);
      dskinf.model_num[i + 1] = (char) (temp & 0xFF);
   }
   readDataReg();/*max sector number tranfered per interrupt*//*word 44*/
   readDataReg();/*double word io*/
   readDataReg();
   readDataReg();
   readDataReg();
   temp = readDataReg();/*word 49*/
```

```
   if(temp & (1 << 9))
   {
      dskinf.is_lba_supp = true;
      enableLba();
   }
   else dskinf.is_lba_supp = false;
//printk("lba:%x\n", (dword)dskinf.is_lba_supp);

   for(i = 0;  i < 10;  i++)/*word 59*/
      readDataReg();

   dskinf.total_sectors = (dword) readDataReg();
   dskinf.total_sectors |= ((dword)readDataReg() << 16);

   for(i = 0;  i < 194;  i++)
      readDataReg();

}

void writeData(void)
{
   size_t i, writeFrom, writeTo;
   word temp;
   byte * tempFirstSector = cmndSt.tempSector;
   byte *tempLastSector = (cmndSt.tempSector + 512);

   while( cmndSt.sectorsCount-- )
   {
      if(cmndSt.curr_sector == cmndSt.firstSector)
         writeFrom = cmndSt.firstByte;
      else
```

```
    writeFrom = 0;

if(cmndSt.curr_sector == cmndSt.lastSector)
    writeTo = cmndSt.lastByte;
else
    writeTo = 511;

for(i = 0;  i < 512;  i+=2 )
{
    temp = 0;

    if(i < writeFrom )
        temp = (word) ( cmndSt.tempSector[i] );
    else if(i > writeTo)
        temp = (word) ( tempLastSector[i]);
    else
    {
        temp = (word) ( cmndSt.ata_buff[cmndSt.buffer_index]);
        cmndSt.buffer_index++;
        cmndSt.bytesReadWriteSucced++;
    }

    if(i + 1 < writeFrom )
        temp |= (word) ( cmndSt.tempSector[i + 1]  << 8 );
    else if(i + 1 > writeTo)
        temp |= (word) ( tempLastSector[i + 1]  << 8 );
    else
    {
        temp |= (word) ( cmndSt.ata_buff[cmndSt.buffer_index]   << 8 );
        cmndSt.buffer_index++;
        cmndSt.bytesReadWriteSucced++;
    }
```

```
            writeDataReg(temp);

        }

        cmndSt.curr_sector++;
    }


    /*
    for(i = 0;  i < 256 * cmndSt.sectorsCount;  i++)
        writeDataReg(cmndSt.ata_buff[cmndSt.buffer_index + i]);

    cmndSt.buffer_index += i;
    */
}

void readBuffer(void)
{
    size_t j;
    word temp;

    if(cmndSt.sectorsCount == 0)
        return;

    if(cmndSt.firstSector == cmndSt.lastSector && cmndSt.curr_sector == cmndSt.firstSector)
    {
        for(j = 0;  j < 512;  j += 2)
        {
            temp = readDataReg();

            if(j >= cmndSt.firstByte  && j <= cmndSt.lastByte)
```

```
        {
            cmndSt.ata_buff[cmndSt.buffer_index] = (byte) (temp & 0x00FF);
            cmndSt.bytesReadWriteSucced++;
            cmndSt.buffer_index++;
        }

        if(j + 1 >= cmndSt.firstByte  && j + 1 <= cmndSt.lastByte)
        {
            cmndSt.ata_buff[cmndSt.buffer_index] = (byte) (temp >> 8);
            cmndSt.bytesReadWriteSucced++;
            cmndSt.buffer_index++;
        }
    }
    return;
}

if(cmndSt.firstSector == cmndSt.curr_sector)
{
    for(j = 0;  j < 512;  j += 2)
    {
        temp = readDataReg();

        if( j >= cmndSt.firstByte )
        {
            cmndSt.ata_buff[cmndSt.buffer_index] = (byte) (temp & 0x00FF);
            cmndSt.bytesReadWriteSucced++;
            cmndSt.buffer_index++;
        }

        if( j + 1 >= cmndSt.firstByte )
        {
            cmndSt.ata_buff[cmndSt.buffer_index] = (byte) (temp >> 8);
```

```
            cmndSt.bytesReadWriteSucced++;
            cmndSt.buffer_index++;
        }
    }
    cmndSt.curr_sector++;
    cmndSt.sectorsCount--;
}

while(cmndSt.sectorsCount)
{
    if(cmndSt.curr_sector == cmndSt.lastSector)
    {
        for(j = 0;  j < 512;  j += 2)
        {
            temp = readDataReg();

            if( j <= cmndSt.lastByte )
            {
                cmndSt.ata_buff[cmndSt.buffer_index] = (byte) (temp & 0x00FF);
                cmndSt.bytesReadWriteSucced++;
                cmndSt.buffer_index++;
            }

            if( j + 1 <= cmndSt.lastByte )
            {
                cmndSt.ata_buff[cmndSt.buffer_index] = (byte) (temp >> 8);
                cmndSt.bytesReadWriteSucced++;
                cmndSt.buffer_index++;
            }
        }
    }
    else    /*if cmndSt.currentSector != cmndSt.lastSector*/
```

```
      {
         for(j = 0;  j < 512;  j += 2)
         {
            temp = readDataReg();

            cmndSt.ata_buff[cmndSt.buffer_index] = (byte) (temp & 0x00FF);
            cmndSt.bytesReadWriteSucced++;
            cmndSt.buffer_index++;

            cmndSt.ata_buff[cmndSt.buffer_index] = (byte) (temp >> 8);
            cmndSt.bytesReadWriteSucced++;
            cmndSt.buffer_index++;
         }
      }



      cmndSt.curr_sector++;
      cmndSt.sectorsCount--;
   }
//printk("test %x\n", (dword) cmndSt.bytesReadWriteSucced);
}

size_t readSectors(dword lba, size_t count, void *buffer)
{
   byte currseccount; /*number of sectors to read */
   size_t readSect;

   if(buffer == NULL)
      return 0;

   if(lba + count > dskinf.total_sectors)
```

```
      count = (size_t) dskinf.total_sectors - lba;

   readSect = count;

   if( !count )
      return 0;

   cmndSt.ata_buff = (byte*) buffer;
   cmndSt.buffer_index = 0;

   while(count > 0)
   {
      setLba(lba);

      if(count >= 256)
      {
         currseccount = 0;
         count -= 256;
         cmndSt.sectorsCount = 256;
      }
      else
      {
         currseccount = count;
         count = 0;
         cmndSt.sectorsCount = currseccount;
      }

      //printk("Alternate Status: %x\n", (dword) readAlternateStatusReg() );
      writeSectorCountReg(currseccount);
//for(size_t i = 0;  i < 40000000; i++)i=i;
      //readAlternateStatusReg();
```

```
        byte status = readStatusReg();
        while(status & 0x80){printk("opp!\n"); for(size_t i = 0;  i < 40000000; i++)i=i; status = readStatusReg();}

        status = readStatusReg();
        while(status & 8){printk("oqq!\n"); for(size_t i = 0;  i < 40000000; i++)i=i; status = readStatusReg();}

        //printk("sending comand....\n");
        //printk("Alternate Status: %x\n", (dword) readAlternateStatusReg() );

        //PicSetIrqMask(14); //??What if interupt trigered before finish
        sendComand(READ_SECTORS_RETRY);



        lba += cmndSt.sectorsCount;
    }

    return (dword) readSect;
}

int writeSectors(dword lba, size_t count, void *buffer)
{
    byte currseccount;

    if(buffer == NULL)
        return -1;

    cmndSt.ata_buff = (byte*) buffer;
    cmndSt.buffer_index = 0;

    while(count > 0)
```

```
   {
      setLba(lba);

      if(count >= 256)
      {
         currseccount = 0;
         count -= 256;
         cmndSt.sectorsCount = 256;
      }
      else
      {
         currseccount = count;
         count = 0;
         cmndSt.sectorsCount = currseccount;
      }

      writeSectorCountReg(currseccount);
      sendComand(WRITE_SECTORS_RETRY);
      while( !(readAlternateStatusReg() & 8) );/*wait for irq*/
      writeData();

      lba += cmndSt.sectorsCount;
   }


   return 0;
}

dword getSize(void)
{
   return dskinf.total_sectors * 512;
}
```

```
dword open(const char* path, const char* rw)
{
   return 1;
}


///return : bytes succesfully readed
size_t read(dword add, size_t count, void *buffer)
{
   size_t i;

   if(count == 0)
      return 0;



   cmndSt.firstByte = add % 512;
   cmndSt.lastByte = (add % 512) + ( (count-1) % 512) % 512; /** (add + count - 1)%512 will
overflow*/

   cmndSt.firstSector = add / 512;
   cmndSt.lastSector = (add / 512) + ((count-1) / 512); /** (add + count)/512 will overflow*/

   cmndSt.bytesReadWriteSucced = 0;
   cmndSt.curr_sector = cmndSt.firstSector;



   wait_interupt = true;
   readSectors(cmndSt.firstSector, cmndSt.lastSector - cmndSt.firstSector + 1, buffer);

   while(wait_interupt);
```

```
   return cmndSt.bytesReadWriteSucced;
}



size_t write(dword add, size_t count, void *buffer, int flags)
{
   size_t i;/**temp*/
   size_t firstByte, lastByte, currSector, firstSector, lastSector;

   if(count == 0)
      return 0;

   firstByte = add % 512;
   lastByte = (add % 512) + ( (count-1) % 512) % 512; /** (add + count - 1)%512 will overflow*/

   firstSector = add / 512;
   lastSector = (add / 512) + ((count-1) / 512); /** (add + count)/512 will overflow*/

   cmndSt.tempSector = NULL;

   if(firstByte != 0)
   {
      cmndSt.tempSector = pageFrameAlloc(1);
      read(firstSector * 512, 512, cmndSt.tempSector);
      for(i = 0;  i < 40000000; i++);/**temp*/
      //printk("temp f: %x\n", cmndSt.tempSector[511]);
   }

   if( lastByte != 511 )
   {
      if(cmndSt.tempSector == NULL)
         cmndSt.tempSector = pageFrameAlloc(1);
```

```
        read(lastSector * 512, 512, cmndSt.tempSector + 512);
        for(i = 0;  i < 40000000; i++);/**temp*/
        //printk("temp l: %x\n", cmndSt.tempSector[511 + 512]);
    }


    ///write begin
    cmndSt.firstByte = firstByte;
    cmndSt.lastByte = lastByte;

    cmndSt.firstSector = firstSector;
    cmndSt.lastSector = lastSector;

    cmndSt.bytesReadWriteSucced = 0;
    cmndSt.curr_sector = cmndSt.firstSector;


    writeSectors(cmndSt.firstSector, cmndSt.lastSector - cmndSt.firstSector + 1, buffer);

    if(cmndSt.tempSector != NULL)
        pageFrameFree((dword)cmndSt.tempSector/PAGE_FRAME_SIZE, 1);



    for(i = 0;  i < 40000000; i++)count = i;/**temp*/
    //wait_interupt = true;
    //while(wait_interupt);

    return cmndSt.bytesReadWriteSucced;

}
```

```
void installDevice(void)
{
   operations op;

   op.getSize = getSize;
   op.read = read;
   op.write = write; //writeSectors;
   op.fopen = open;
   op.fseek = NULL;

   initDisk();
   addDevice(dskinf.model_num, dskinf.serial_num, STORAGE, &op);
}

static void errorHandler(void)
{
   byte error = readErrorReg();

   switch(cmndSt.lastCommand){
      case READ_SECTORS_RETRY:
      case READ_SECTORS:
         ///if(error == x ) total_bytes_read = 0;

         break;

   }
}

INTR_HNDL void ide_handler(struct interrupt_frame *frame)
{
   byte status = readAlternateStatusReg();
```

```
   if(status & 1)
   {
      printk("IDE:Error occured:\nError reg: %x Status: %x", (dword) readErrorReg(), (dword)
status);
      errorHandler();
   }
   else /// if no errors
   {

      switch(cmndSt.lastCommand){
         case IDENTIFY_DRIVE:
            setDskInf();
            //printk("lbas %x\n",  (dword)dskinf.total_sectors);
            cmndSt.identifyDriveComlete = true;
            break;
         case READ_SECTORS_RETRY:
         case READ_SECTORS:
            //printk("read_sectors: %x\n", (dword) readSectorCountReg());
            readBuffer();
            break;
         case WRITE_SECTORS_RETRY:
            //printk("trasfer completed!!!\n");
            break;
      }
   }

   readStatusReg();
   PicEoi(PRIMARY_ATA_IRQ);/*Check correct irq from array and selected drive*/
   wait_interupt = false;
}
```

Αρχείο:interupts.c

```
#include "interupts.h"
#include "keyboard.h"
#include "procScheduler.h"
#include "exeptions.h"
```

```
#include "segments.h"
#include "exportedSymbols.h"

struct interuptGate idt[256];

void sys_call_hndlr(void);
void pitHndlr(void);

void lidt(void);
void installExcHandlers(void);
void installExcHandlers(void);   /**??*/

void initIdt(void)
{
   /*Initialize idt with the exception hanlers*/
   installExcHandlers();

   /*load idt into register*/
   lidt();
}

INTR_HNDL void spuriousIntHandler(struct interrupt_frame *frame)
{
   //__asm volatile("cli;");
   //printk("32423423!@#");
   //while(1);
}

void addIntGate(size_t index, word cs, dword offset, byte flags)
{

   idt[index].offsetLow16 = (word) (offset & 0xFFFF);
```

```
   idt[index].segmentSelector = cs;
   idt[index].zero = (byte)0;
   idt[index].typeAttr = flags;
   idt[index].offsetHigh16 = (word) (offset >> 16);
}

void installExcHandlers(void)
{
   addIntGate(0, CS_PL0, (dword) divZeroEx,  INT_GATE | DPL0);  //flags = 8E
   addIntGate(1, CS_PL0, (dword) debugEx,  INT_GATE | DPL0);
   addIntGate(3, CS_PL0, (dword) breakpointEx,  INT_GATE | DPL3);   //flags = EE
   addIntGate(4, CS_PL0, (dword) overflowEx,  INT_GATE | DPL3);
   addIntGate(5, CS_PL0, (dword) boundEx,  INT_GATE | DPL3);
   addIntGate(6, CS_PL0, (dword) invalodOpCodeEx,  INT_GATE | DPL0);
   addIntGate(7, CS_PL0, (dword) deviceNotAvailEx,  INT_GATE | DPL0);
   addIntGate(8, CS_PL0, (dword) doubleFaultEx,  INT_GATE | DPL0);
   addIntGate(9, CS_PL0, (dword) coprocessorOverrunEx,  INT_GATE | DPL0);
   addIntGate(10, CS_PL0, (dword) tssEx,  INT_GATE | DPL0);
   addIntGate(11, CS_PL0, (dword) segmentNotPresentEx,  INT_GATE | DPL0);
   addIntGate(12, CS_PL0, (dword) StackFaultEx,  INT_GATE | DPL0);
   addIntGate(13, CS_PL0, (dword) generalProtectionFaultEx,  INT_GATE | DPL0);
   addIntGate(14, CS_PL0, (dword) PageFaultEx,  INT_GATE | DPL0);
   addIntGate(16, CS_PL0, (dword) x87FPUErrorEx,  INT_GATE | DPL0);
   addIntGate(17, CS_PL0, (dword) AligmentCheckEx,  INT_GATE | DPL0);
   addIntGate(18, CS_PL0, (dword) MachineCheckEx,  INT_GATE | DPL0);
   addIntGate(19, CS_PL0, (dword) SIMDFloatPointEx,  INT_GATE | DPL0);

addIntGate(39, CS_PL0, (dword) spuriousIntHandler,  INT_GATE | DPL0);

   addIntGate(32, CS_PL0, (dword) pitHndlr,  INT_GATE | DPL0);
   //addIntGate(33, CS_PL0, (dword) keyboardHandler,  INT_GATE | DPL0);  //?? protect from gpf
if hardware hanlder is not installed
   addIntGate(255, CS_PL0, (dword) sys_call_hndlr,  TRAP_GATE | DPL3);
```

```
}

void lidt(void)
{
   interuptDescTableReg.limit = 0x1000;
   interuptDescTableReg.base = (dword) idt;

   __asm  volatile ("lidt %0;" : : "m"(interuptDescTableReg));
}
```

Αρχείο: ioblock.c

```
#include "ioblock.h"

extern pcbNode *currentProcces, *readyProccesQueue, *readyProccesQueueTail;
pcbNode *ioBlockedProccesQueue;
```

```
void ioblock(void)
{

   if(currentProcces != NULL && currentProcces == readyProccesQueue)
   {
      if(readyProccesQueueTail == readyProccesQueue)
         readyProccesQueueTail = NULL;

      readyProccesQueue = readyProccesQueue->next;

      if(ioBlockedProccesQueue == NULL)
         ioBlockedProccesQueue = currentProcces;
      else
      {
         currentProcces->next = ioBlockedProccesQueue;
         ioBlockedProccesQueue = currentProcces;
      }

      currentProcces = NULL;
      __asm volatile("int $32");
   }
}

void iounblock(pcbNode *proc)
{
   pcbNode *curr = ioBlockedProccesQueue, *prev = NULL;

   if(currentProcces != NULL && currentProcces == readyProccesQueue)
   {
      while(curr != NULL && curr != proc)
      {
```

```
            prev = curr;
            curr = curr->next;
        }

        if(curr == NULL)
            return;

        if(prev == NULL)
            ioBlockedProccesQueue = ioBlockedProccesQueue->next;
        else
            prev->next = curr->next;

        readyProccesQueueTail->next = curr;
        readyProccesQueueTail = curr->next;
        readyProccesQueueTail->next = NULL;

    }
}
```

Αρχείο: keyboard.c

```
#include "keyboard.h"
#include "io.h"
#include "vga.h"
#include "pic.h"
```

```
const byte scanToAscii[] = {'\0', '\0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '-', '=', '\0', '\0', 'q', 'w', 'e', 'r', 't',
                'y', 'u', 'i', 'o', 'p', '[', ']', '\0', '\0', 'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l',
                ';', '\'', '\0', '\0', '\\', 'z', 'x', 'c', 'v', 'b', 'n', 'm', ',', '.', '/', '\0', '\0', '\0', '\0', '\0', '\0',
                '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0',
                '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0',
                '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0',
                '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0',
                '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0',
                '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0',
                '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0',
                '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0',
                '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0',
                '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0'};

void keyboardHandler(void)
{
   byte scanCode, ascii;

   INB(scanCode, KEYBOARD_DATA_BUFFER);

   ascii = scanToAscii[scanCode];

   if(KEY_PRESSED(scanCode))
      printChar(ascii, 0x0F);
```

```
    PicEoi(ISA_KEYBOARD_IRQ);

    __asm volatile("sti;");
    __asm volatile("leave;");
    __asm volatile("iret;");
}
```

Αρχείο:main.c

```
#include "type.h"
#include "registers.h"
#include "vga.h"
#include "io.h"
```

```
#include "exeptions.h"
#include "interupts.h"
#include "stdio.h"     ///???
#include "keyboard.h"  ///???
#include "procScheduler.h"
#include "memory.h"
#include "criticalError.h"
#include "pic.h"
#include "version.h"
#include "kernelHeap.h"
#include "sysCallTable.h"
//#include "sysCallLib.h"
#include "stdiok.h"
#include "ide.h"
#include "createProcces.h"
#include "device.h"
#include "string.h"
#include "volume.h"
#include "exportedSymbols.h"
#include "gui.h"

extern dword debug;
void _main() __asm ("_main");  ///for entry point in gnu linker we define _main at 0x500
void k_main();

void sysWait(void);

/*proc1() and proc2() are procceses that run at the same time
*with task switching.
proc1() print a message on the screen while proc2() blinks keyboard leds
*/
void proc1(void);
```

```
void proc2(void);
void proc3(void);
void initIde(void);

void delay();

/*_main function called from the bootloader
 *after switching to protected mode.
 *Gnu linker must put this function where the bootloader makes far jump(0x500)
 */
void _main()
{
   /*Initialize segment registers
    *at the third entry of the gdt
    *In flat memory model all segment registers use the same desciptor.
    *Cs register initialized in bootloader(whith a far jump) at the second entry of gdt(0x08)
    *The 3 lsb must be zero (privilage level and table indicator)
    *  |index at gdt | ti| rpl
    *  |0000000000010| 0 | 00b = 0x10
    */
   SET_DS(0x10);
   SET_ES(0x10);
   SET_FS(0x10);
   SET_GS(0x10);
   SET_SS(0x10);

   /*Initialize stack
   *Bios memory map mark erea 0x00007E00 - 0x0007FFFF as
   *guaranteed free for use. Stack grows downwards hence ebp and esp
   *initiazed at 0x0007FFFF
   */
   SET_EBP(0x4FFFFC);///!!!??stack must change address!!!
```

```
   SET_ESP(0x4FFFFC);

   k_main();

}

void k_main()
{

   OUTB(0x36, 0x43);
   OUTB(166, 0x40);
   OUTB(11, 0x40);

   /*
   __asm volatile("movb $54, %al;"
                  "outb  %al, $67;"
                  "movw $11931, %ax;"
                  "movw $11931, %ax;"
                  "movw $11931, %ax;"
                  "movw $11931, %ax;"
                  );/*



   /*Clear screen from bios message*/
   init_vga(); //clearScreen();

   /*initialize idt*/
   printk("Initializing idt...\n");

   initIdt();
```

```
   /*Remap pic irqs*/
   printk("Remmaping pic at 0x%x and
0x%x...\n",(dword)DEFAULT_MASTER_PIC_VECTOR_OFFSET,
(dword)DEFAULT_SLAVE_PIC_VECTOR_OFFSET);
   if(remapIrq((byte)DEFAULT_MASTER_PIC_VECTOR_OFFSET,
(byte)DEFAULT_SLAVE_PIC_VECTOR_OFFSET) == -1)
   {
      printk("Unkown error: irqs not remaped...\nPlease reestart the system\n");
      while(1); /*Stop system initialization*/
   }

   /*Create a bitmap that describe if a page frame is used or not
    *With that bitmap we can mark page frames as free or used and we are able to allocate
memory
   */
   printk("Initializing memory map...\n");
   initMemoryMap();

   /*initialize kernel's heap*/
   printk("initialze kernel's heap...\n");
   initKernelHeap();

   /*load tss*/
   memset((void*)54, 0, 0x67);    ///?? at 0x68 size of tss??
https://wiki.osdev.org/Task_State_Segment
   SET_EAX(0x2b);
   __asm volatile("ltr %ax;");
   (*(byte*)0x3E) = 0x10; /*initialize ss0 at tss*/

   /*initialize process array*/
   printk("Initializing process table...\n");
   initProc();
   initVideo();
```

```
   //clearScreen();


   /*Configure programmable interrupt controller (PIC)
    *to send only keyboard, ata and pit hardware interupts to processor
   */
   OUTB(0xFB, 0x21);
   OUTB(0xFF, 0xA1);//OUTB(0xFF, 0xA1);

   initVolumes();
   initIde();
   /*call volume search*/


   printk("Initializing expored symbols table...\n");
   initExportedSymbols();


   clearScreen();
   __asm volatile("sti;");


//    addDriver("A:/KEYBOARD.DRV");
   //addDriver("A:/MOUSE.DRV");
 //   addDriver("A:/OUT.RUN");

   //addDriver("A:/KBRDCLI.RUN");
   /**createProcces("A:/GETCHAR.RUN");   */  ///working
   SccreateProcces("A:/SRV.RUN",  "");
   //createProcces("A:/GETCHSC.RUN");
   SccreateProcces("A:/CMD.RUN", "");
   SccreateProcces("A:/START.RUN",  "");
```

```
   //createProcces("A:/MAIN.RUN");
   //createProcces("A:/TEST2.RUN");
   //createProcces("A:/TEST3.RUN");
   clearScreen();
   __asm volatile("cli;");

   PicClearIrqMask(1);
   PicClearIrqMask(0);
   //PicClearIrqMask(2);
   PicClearIrqMask(12);
   //PicClearIrqMask(14);

   SET_DS(0x1B);
   SET_ES(0x1B);
   SET_FS(0x1B);
   SET_GS(0x1B);




   /*Enable interupts and wait pit interupt for task switching*/
   __asm volatile("sti;");
   here:goto here;
   __asm volatile("hlt;");

}

extern blockedProccesQueue, ioBlockedProccesQueue;

void sysWait(void)
{
```

```
   while(1)
   {
      //printk("wait, %x, %x ", (dword) blockedProccesQueue, (dword) ioBlockedProccesQueue );
      __asm volatile("sti;");
      __asm volatile("hlt;");
   }
}

void initIde(void)
{
   PicClearIrqMask(14);
   __asm volatile("sti;");

   selectDrive(MASTER);
   //initDisk();
   installDevice();

   __asm volatile("cli;");

}
```

Αρχείο: memory.c

```
#include "memory.h"
#include "stdio.h"
#include "stdiok.h"
```

```
#define PROCCES_VER 2
#if  PROCCES_VER == 1
extern procStack proccesStack;
#endif // PROCCES_VER
#if  PROCCES_VER == 2
extern pcbNode *readyProccesQueue, *currentProcces;
#endif // PROCCES_VER


memoryMapEntry *memoryMap;  /*Array created with bios function and describes system memory. Evry entry describe a region.*/
dword biosMemoryMapEntriesCount;    /*Number of entries in memory map*/
dword memory_size;  /*total memory installed in system in Bytes*/

dword createMemoryPageFrameBitmap();
dword findRegion(size_t count);
bool regionIsFree(dword basePage, size_t countPages);
#if PROCCES_VER == 1
catalogEntry* realocateProccessCatalog(catalogEntry *memCat, size_t newSize );
#elif PROCCES_VER == 2
pageFrameRegionEntry* realocateProccessCatalog(pageFrameRegionEntry *memCat, size_t newSize );
#endif // PROCCES_VER

void initMemoryMap(void)
{

   memoryMap = BIOS_MEMORY_MAP_ADDRESS;
   biosMemoryMapEntriesCount = *MEMORY_MAP_ENTRIES_COUNT;
   memory_size =(dword) memoryMap[biosMemoryMapEntriesCount - 1].baseAddress + memoryMap[biosMemoryMapEntriesCount - 1].regionLength;

   createMemoryPageFrameBitmap();
```

```
   //printf("Total memory(Byte): %x\n", (dword)memory_size);
}

dword getMemorySize(void)
{

   return memory_size;
}

void printMemoryMap(void)
{
   size_t i;

   printk("BASE     REG LENTH  TYPE\n");
   for(i = 0;  i < (size_t) biosMemoryMapEntriesCount; i++ )
   {
      printk("%x | %x | %x\n", (dword)memoryMap[i].baseAddress, (dword)
memoryMap[i].regionLength, (dword) memoryMap[i].regionType);
   }
}

dword createMemoryPageFrameBitmap()
{
   size_t i = 0;

   memset(PAGE_FRAME_BITMAP_BASE_ADD, 0, (size_t) (memory_size/PAGE_FRAME_SIZE/8) );
   for(i = 0;  i < biosMemoryMapEntriesCount;  i++)
   {
      if(memoryMap->regionType != REG_TYPE_FREE)
      {
         size_t pageCount;
```

```
            pageCount = memoryMap->regionLength / PAGE_FRAME_SIZE;
            if( memoryMap->regionLength % PAGE_FRAME_SIZE ) pageCount++;
            if( ((memoryMap->baseAddress % PAGE_FRAME_SIZE) + ( memoryMap->regionLength %
PAGE_FRAME_SIZE)) > PAGE_FRAME_SIZE)pageCount++;
            ReservePageFrames( GET_PAGE_FRAME_FROM_ADDRESS(memoryMap->baseAddress),
pageCount);

        }
    }

    ReservePageFrames(0, 5242880/4096);   /**First 5Mb are reserved for the system*/


    return 0;
}

void* pageFrameAlloc(size_t frameCnt)
{
    dword startPage;

    if( (startPage = findRegion(frameCnt)) == 0)
        return (void*) 0;

    ReservePageFrames(startPage, frameCnt);

    return (void*)(startPage * (dword)PAGE_FRAME_SIZE );
}

void* pageFrameAllocBase(size_t frameCnt, dword basePage)
{
    if( !regionIsFree(basePage, frameCnt) )
        return NULL;
```

```
   ReservePageFrames(basePage, frameCnt);

   return (void*)(basePage * (dword)PAGE_FRAME_SIZE );
}

void pageFrameFree(dword basePage, size_t count)
{
   ReleasePageFrames(basePage, count);
}

dword findRegion(size_t count)
{
   dword pageNum = 0, pageCount = GET_PAGE_FRAME_FROM_ADDRESS(memory_size) +
1;///total pages   ??+1
   size_t regionSize;

   while(pageNum + count < pageCount)
   {
      regionSize = 0;
      while( (*(byte*)(((dword)PAGE_FRAME_BITMAP_BASE_ADD) + (pageNum + regionSize)/8) &
(128 >> ((pageNum + regionSize) % 8) ))  == 0)
      {
         regionSize++;
         if(regionSize == count) return pageNum;
      }

      pageNum += (regionSize + 1);
   }

   return 0;
}

bool regionIsFree(dword basePage, size_t countPages)
```

```
{
   dword sysTotalPages = GET_PAGE_FRAME_FROM_ADDRESS(memory_size) - 1;  ///global??
   dword currPage = basePage, lastPage = basePage + countPages - 1;

   if(lastPage  >  sysTotalPages)
      return false;

   while(currPage <= lastPage)
   {
      if(GET_PAGE_STATUS(currPage) != PAGE_IS_FREE )
         return false;

      currPage++;
   }

   return true;
}




void ReservePageFrames(dword basePgFrame, size_t pgFrameCount)
{
   size_t i;
   byte *pBitmap = (byte*) ((dword) PAGE_FRAME_BITMAP_BASE_ADD + (dword)
(basePgFrame/ (dword)8) );



   for(i = 0;  i < pgFrameCount;  i++)
   {
      *pBitmap |= (byte) 128 >> basePgFrame++ % (dword)8;

      if((basePgFrame) % 8 == 0)pBitmap++;
   }
```

```
}

void ReleasePageFrames(dword basePgFrame, size_t pgFrameCount)
{
   size_t i;
   byte *pBitmap = (byte*) ((dword) PAGE_FRAME_BITMAP_BASE_ADD + (dword)
(basePgFrame/ (dword)8) );


   for(i = 0;  i < pgFrameCount;  i++)
   {
      *pBitmap &= ~ ((byte) 128 >> basePgFrame++ % (dword)8 );

      if(basePgFrame % 8 == 0)pBitmap++;
   }
}
#if  PROCCES_VER == 1
catalogEntry* createProccessCatalog(void)
{
   catalogEntry *memCat = NULL;

   memCat = (catalogEntry*) pageFrameAlloc(1);
   if(memCat == NULL)
      return NULL;

   memCat[0].base = (void*) memCat;
   memCat[0].length = 1;

   return memCat;
}

void* ScAllocatePhysicalPages(dword pid, size_t length)
{
```

```
   void *tempRealoc = NULL, *tempMem = NULL;

   if((tempMem = (void*) pageFrameAlloc(length)) == NULL )
      return NULL;



   if( (proccesStack.procces[pid].memCat.entrCnt + 1) * sizeof(catalogEntry) >
      proccesStack.procces[pid].memCat.ctlgEntr[0].length * PAGE_FRAME_SIZE -
sizeof(catalogEntry))
   {
      if( (tempRealoc =
         realocateProccessCatalog(proccesStack.procces[pid].memCat.ctlgEntr,
                        proccesStack.procces[pid].memCat.ctlgEntr[0].length * 2)) == NULL)
      {
         pageFrameFree((dword)tempMem / PAGE_FRAME_SIZE, length);
         return NULL;
      }

      proccesStack.procces[pid].memCat.ctlgEntr = tempRealoc;
   }

   proccesStack.procces[pid].memCat.entrCnt++;
   proccesStack.procces[pid].memCat.ctlgEntr[proccesStack.procces[pid].memCat.entrCnt -
1].length = length;

   return proccesStack.procces[pid].memCat.ctlgEntr[proccesStack.procces[pid].memCat.entrCnt
- 1].base = tempMem;
}

catalogEntry* realocateProccessCatalog(catalogEntry *memCat, size_t newSize )
{
   catalogEntry *ptr = NULL;
```

```
   if((ptr = (catalogEntry*) pageFrameAlloc(newSize)) == NULL)
      return NULL;

   memcpy((byte*) memCat, (byte*) ptr, memCat[0].length * PAGE_FRAME_SIZE);
   ptr[0].base = ptr;
   ptr[0].length = newSize;

   pageFrameFree((dword)memCat / PAGE_FRAME_SIZE, memCat[0].length);

   return ptr;
}



int ScFreePhysicalPages(void* ptr, dword pid)///!pid parametre must eliminated
{
   size_t i, lastRecord;

   /*Check if ptr record exist in catalog*/
   if(proccesStack.procces[pid].memCat.entrCnt > 2)
   {
      for(i = 2;  i < proccesStack.procces[pid].memCat.entrCnt; i++)
      {
         if(proccesStack.procces[pid].memCat.ctlgEntr[i].base == ptr)
         {
            ///delete entry and decrementnt entrCnt
            ///replace last entry with entry which is going to be deleted
            proccesStack.procces[pid].memCat.entrCnt--;
            pageFrameFree((dword) ptr / PAGE_FRAME_SIZE, (size_t)
proccesStack.procces[pid].memCat.ctlgEntr[i].length);

            lastRecord = proccesStack.procces[pid].memCat.entrCnt;
            proccesStack.procces[pid].memCat.ctlgEntr[i].base =
proccesStack.procces[pid].memCat.ctlgEntr[lastRecord].base;
```

```
            proccesStack.procces[pid].memCat.ctlgEntr[i].length =
proccesStack.procces[pid].memCat.ctlgEntr[lastRecord].length;
            proccesStack.procces[pid].memCat.ctlgEntr[i].type =
proccesStack.procces[pid].memCat.ctlgEntr[lastRecord].type;
            return 0;
         }
      }
   }

   return -1;
}

void destroyProccesCatalog(dword pid)
{
   size_t i;

   for(i = 1;  i < proccesStack.procces[pid].memCat.entrCnt; i++)
   {
      pageFrameFree((dword) proccesStack.procces[pid].memCat.ctlgEntr[i].base /
PAGE_FRAME_SIZE,
               (size_t) proccesStack.procces[pid].memCat.ctlgEntr[i].length);
   }

   pageFrameFree((dword) proccesStack.procces[pid].memCat.ctlgEntr[0].base /
PAGE_FRAME_SIZE,
            (size_t) proccesStack.procces[pid].memCat.ctlgEntr[i].length);
}
#endif // PROCCES_VER
#if  PROCCES_VER == 2
pageFrameRegionEntry* createProccessCatalog(void)
{
   pageFrameRegionEntry *pageFrameCat = NULL;

   pageFrameCat = (pageFrameRegionEntry*) pageFrameAlloc(1);
```

```
   if(pageFrameCat == NULL)
      return NULL;

   pageFrameCat[0].base = (void*) pageFrameCat;
   pageFrameCat[0].length = 1;



   return pageFrameCat;
}

static void* allocatePhysicalPages(size_t length, dword base_address, dword mode, dword
protection)
{
   if(mode == ALLOC_MODE_ANY_ADD)
      return pageFrameAlloc(length);

   if(mode == ALLOC_MODE_FIXED_ADD)
      return  pageFrameAllocBase(length, base_address);

   return NULL;
}

/*Param: length: number of page frames to allocate*/
void* ScMemAlloc(size_t length, dword base_address, dword mode, dword protection)
{
   void *tempRealoc = NULL, *tempMem = NULL;

   if((tempMem = allocatePhysicalPages(length, base_address, mode, protection)) == NULL )
      return NULL;

   /*check if free space for new entry exist*/
   if( currentProcces->pageCat.count * sizeof(pageFrameRegionEntry) >= (currentProcces-
>pageCat.catalog[0].length * PAGE_FRAME_SIZE) )
```

```
   {
      ///temp realoc is not *pageFrameRegionEntry
      if( (tempRealoc = realocateProccessCatalog(currentProcces->pageCat.catalog,
currentProcces->pageCat.count + 10)) == NULL)  ///??dynamic not +100
      {
         pageFrameFree((dword)tempMem / PAGE_FRAME_SIZE, length);
         return NULL;
      }

      currentProcces->pageCat.catalog = tempRealoc;
   }

   currentProcces->pageCat.count++;
   currentProcces->pageCat.catalog[currentProcces->pageCat.count - 1].length = length;

   return currentProcces->pageCat.catalog[currentProcces->pageCat.count - 1].base =
tempMem;
}

pageFrameRegionEntry* realocateProccessCatalog(pageFrameRegionEntry *memCat, size_t
newSize )
{
   pageFrameRegionEntry *ptr = NULL;

   if((ptr = (pageFrameRegionEntry*) pageFrameAlloc(newSize)) == NULL)
      return NULL;

   memcpy((byte*) memCat, (byte*) ptr, memCat[0].length * PAGE_FRAME_SIZE);
   ptr[0].base = ptr;
   ptr[0].length = newSize;

   pageFrameFree((dword)memCat / PAGE_FRAME_SIZE, memCat[0].length);
```

```
   return ptr;
}

int ScFreePhysicalPages(void* ptr)///!pid parametre must eliminated
{
   size_t i, lastRecord;

   /*Check if ptr record exist in catalog*/
   if(currentProcces->pageCat.count > 2)
   {
      for(i = 2;  i < currentProcces->pageCat.count; i++)
      {
         if(currentProcces->pageCat.catalog[i].base == ptr)
         {
            ///delete entry and decremennt entrCnt
            ///replace last entry with entry which is going to be deleted
            currentProcces->pageCat.count--;
            pageFrameFree((dword) ptr / PAGE_FRAME_SIZE,  currentProcces-
>pageCat.catalog[i].length);

            lastRecord = currentProcces->pageCat.count;
            currentProcces->pageCat.catalog[i].base = currentProcces-
>pageCat.catalog[lastRecord].base;
            currentProcces->pageCat.catalog[i].length = currentProcces-
>pageCat.catalog[lastRecord].length;
            currentProcces->pageCat.catalog[i].type = currentProcces-
>pageCat.catalog[lastRecord].type;
            return 0;
         }
      }
   }

   return -1;
}
```

```
void destroyProccesCatalog()
{
   size_t i;

   for(i = 1;  i <currentProcces->pageCat.count; i++)
      pageFrameFree((dword) currentProcces->pageCat.catalog[i].base /
PAGE_FRAME_SIZE,currentProcces->pageCat.catalog[i].length);

   pageFrameFree((dword) currentProcces->pageCat.catalog[0].base / PAGE_FRAME_SIZE,
currentProcces->pageCat.catalog[0].length);
}

#endif // PROCCES_VER

size_t countFreePhysicalPages(void)
{
   size_t i, cnt = 0;
   byte *pBitmap = (byte *) PAGE_FRAME_BITMAP_BASE_ADD;

   for(i = 0;  i < memory_size / PAGE_FRAME_SIZE;  i++)
   {
      if( (pBitmap[i/8] & (128 >> (i % 8))) == 0)cnt++;

   }
   return cnt;
}
```

Αρχείο: pe.c

```
#include "pe.h"
#include "vfs.h"
#include "memory.h"
#include "stdiok.h"
```

```
static void printProcProp(peHeader *peH);
static bool loadSectionsToMem(peHeader *peH, byte *ibuffer, int type);

void loadProcFromDisk(const char *exepath, dword *entry, int type)
{
   peHeader peH;
   dword image;
   size_t fsize, frealsize;
   byte *ibuffer = pageFrameAlloc(50);  ///?? free memory
printk("pe\n");

   if( NULL == ibuffer)
   {
      printk("Failed to allocate memmory...\n");
      return;
   }

   if(vffileExists(exepath, NULL, &frealsize) != 1)
   {
      printk("File does not exist\n");
      return;
   }

   image = vfopen(exepath, "r", &fsize);
   if(image == 0xFFFFFFFF)
   {
      printk("File don't open...\n");
      return;
   }

   fsize = vfread( exepath, &image, 0, frealsize, ibuffer);
```

```
vfclose(image);

if(fsize != frealsize)
{
   printk("Error could not real file correct.\n");
   return;
}

peH.peSign = (dword) (* (word*)&ibuffer[0]);
peH.peHdrOff = (size_t) (* (dword*)&ibuffer[0x3c]);
peH.peHdrSign = (dword) (* (dword*)&ibuffer[peH.peHdrOff]);
peH.SecCnt = (dword) (* (word*)&ibuffer[peH.peHdrOff + 6]);
peH.OpHdrSz = (size_t) (* (word*)&ibuffer[peH.peHdrOff + 20]);

if(!peH.OpHdrSz)
{
   printk("Optional header does not exist...\n");
   return;
}

peH.OpHdrOff = peH.peHdrOff + PE_COFF_HEADER_SZ;
peH.magicNum = (dword) (* (word*)&ibuffer[peH.OpHdrOff]);
peH.codeSz = (dword) (* (dword*)&ibuffer[peH.OpHdrOff + 4]);
peH.initDSz = (dword) (* (dword*)&ibuffer[peH.OpHdrOff + 8]);
peH.uninDSz = (dword) (* (dword*)&ibuffer[peH.OpHdrOff + 12]);
peH.rvaEntP = (dword) (* (dword*)&ibuffer[peH.OpHdrOff + 16]);
peH.rvaCodeBase = (dword) (* (dword*)&ibuffer[peH.OpHdrOff + 20]);
peH.rvaDataBase = (dword) (* (dword*)&ibuffer[peH.OpHdrOff + 24]);
peH.imagebase = (dword) (* (dword*)&ibuffer[peH.OpHdrOff + 28]);
peH.SecTblOff = peH.OpHdrOff + peH.OpHdrSz;
```

```
   if(PE_FORMAT_PE32 != peH.magicNum )
   {
      printk("Format is not PE32...\n");
      return;
   }

   if( !loadSectionsToMem(&peH, ibuffer, type) )
   {
      printk("Loading sections failed!\n");
      return;
   }



   *entry = peH.imagebase + peH.rvaEntP;
   printProcProp(&peH);

}

static bool loadSectionsToMem(peHeader *peH, byte *ibuffer, int type)
{
   size_t i, j;
   dword secFlags, secOffset, secSize, virtualAdd;
   byte **sectionAdd;

   sectionAdd = kmalloc(peH->SecCnt * sizeof(byte*) );  /**??free memory*/

   for(i = 0;  i < peH->SecCnt; i++)
   {
      secFlags = (dword) (* (dword*)&ibuffer[peH->SecTblOff + 36]);
      if( (secFlags & 0x20) || (secFlags & 0x40) || (secFlags & 0x80) )   /**code or data segment*/
      {
         //peH->codeSecOff = (dword) (* (dword*)&ibuffer[peH->SecTblOff + 20]);
```

```
        secOffset = (dword) (* (dword*)&ibuffer[peH->SecTblOff + 20]);
        secSize = (dword) (* (dword*)&ibuffer[peH->SecTblOff + 16]);
        virtualAdd = (dword) (* (dword*)&ibuffer[peH->SecTblOff + 12]);

        if(type == 1)
           sectionAdd[i] = pageFrameAllocBase(secSize / 4096+4, (virtualAdd+peH-
>imagebase)/4096 );   ///??Free memory  ??not in proccess catalog
        else if(!type)
           sectionAdd[i] = ScMemAlloc((secSize /4096)+1, (virtualAdd+peH->imagebase)/4096,
ALLOC_MODE_FIXED_ADD, 0);


        if(sectionAdd[i] == NULL)
        {
           printProcProp(peH);
           printk("section's memory allocation faild(section size=%x page=%x)...\n",  secSize,
(virtualAdd+peH->imagebase)/4096  );
           while(1);
           return false;
        }

        copyBytes(sectionAdd[i], (byte*) &ibuffer[secOffset], secSize );
     }
     else
        sectionAdd[i] = NULL;


     peH->SecTblOff += 40;
  }

  return true;
}
```

```
static void printProcProp(peHeader *peH)
{
   //size_t  i;

   printk("Signature of PE is: %x\n", peH->peSign );
   printk("Pointer to PE Header: %x\n",(dword) peH->peHdrOff );
   printk("PE Header signature is: %x\n", peH->peHdrSign );
   printk("Number of sections is: %x\n", peH->SecCnt );
   printk("SizeOfOptionalHeader is %x\n", (dword) peH->OpHdrSz );

   if(!peH->OpHdrSz)
   {
      printk("Optional header does not exist...\n");
      return;
   }

   printk("Magic number is %x\n", peH->magicNum );

   if(PE_FORMAT_PE32 != peH->magicNum )
   {
      printk("Format is not PE32...\n");
      return;
   }

   printk("Size of code= %x\n",     (dword) peH->codeSz );
   printk("Size of init data %x\n", (dword) peH->initDSz );
   printk("Size of unin data %x\n", (dword) peH->uninDSz );
   printk("RVA entry point %x\n",    peH->rvaEntP );
   printk("RVA base of code %x\n",   peH->rvaCodeBase );
   printk("RVA base of data %x\n",   peH->rvaDataBase );
   printk("Image base %x\n",         peH->imagebase );
```

```
   //printk("Image size = %x\n", fsize);


   printk("Printing sections...\n");
   printk("SectionN| VirtSize | VirtAddr | File Off | SzofRawD |\n");


/*
   for(i = 0;  i < peH->SecCnt; i++)
   {
      printk("%c%c%c%c", (byte) (* (byte*)&ibuffer[SecTblOff]), (byte) (*
(byte*)&ibuffer[SecTblOff+1]), (byte) (* (byte*)&ibuffer[SecTblOff+2]), (byte) (*
(byte*)&ibuffer[SecTblOff+3])   );
      printk("%c%c%c%c| ", (byte) (* (byte*)&ibuffer[SecTblOff+4]), (byte) (*
(byte*)&ibuffer[SecTblOff+5]), (byte) (* (byte*)&ibuffer[SecTblOff+6]), (byte) (*
(byte*)&ibuffer[SecTblOff+7])   );
      printk("%x | %x | %x | %x\n", (dword) (* (dword*)&ibuffer[SecTblOff + 8]), (dword) (*
(dword*)&ibuffer[SecTblOff + 12]), (dword) (* (dword*)&ibuffer[SecTblOff + 20]), (dword) (*
(dword*)&ibuffer[SecTblOff + 16]) );
      SecTblOff += 40;
   }
*/
}
```

Αρχείο: pic.c

```
#include "pic.h"
#include "type.h"
#include "io.h"
```

```
s32bit remapIrq(byte mstVecOff, byte slvVecOff)
{
   ///If vector offset are not 8 bit aligned return error
   if( ((mstVecOff | slvVecOff) & 0x7) != 0 )
      return -1;

   OUTB(ICW1_INIT+ICW1_ICW4, PIC1_COMMAND);
   OUTB(ICW1_INIT+ICW1_ICW4, PIC2_COMMAND);

   OUTB(mstVecOff, PIC1_DATA);
   OUTB(slvVecOff, PIC2_DATA);

   OUTB(0x4, PIC1_DATA);
   OUTB(0x2, PIC2_DATA);

   OUTB(ICW4_8086, PIC1_DATA);
   OUTB(ICW4_8086, PIC2_DATA);

   return 0;
}

/*Send to the pic end of interrupt message*/
KEEP_REGS void PicEoi(byte irq)
{
   if(irq > 8)
      OUTB(PIC_EOI, PIC2_COMMAND);

   OUTB(PIC_EOI, PIC1_COMMAND);
}

/*Mask an Irq line*/
void PicSetIrqMask(byte irqLine)
```

```
{
   byte mask;
   byte port;

   if(irqLine < 8)
   {
      INB(mask, PIC1_DATA);
      port = PIC1_DATA;
   }
   else
   {
      INB(mask, PIC2_DATA);
      irqLine -= 8;
      port = PIC2_DATA;
   }

   mask |= (1 << irqLine);
   OUTB(mask, port);

}

void PicClearIrqMask(byte irqLine)
{
   byte mask;
   byte port;

   if(irqLine < 8)
   {
      INB(mask, PIC1_DATA);
      port = PIC1_DATA;
   }
   else
```

```
    {
        INB(mask, PIC2_DATA);
        irqLine -= 8;
        port = PIC2_DATA;
    }

    mask &= ~(1 << irqLine);
    OUTB(mask, port);

}

byte PicGetIrqState(byte irqLine)
{
    byte mask;
    byte port;

    if(irqLine < 8)
    {
        INB(mask, PIC1_DATA);
        port = PIC1_DATA;
    }
    else
    {
        INB(mask, PIC2_DATA);
        irqLine -= 8;
        port = PIC2_DATA;
    }

    return (mask >> irqLine) & 1;

}
```

```
byte isIrqPitEnable(void)
{
   return !PicGetIrqState(ISA_PIT_IRQ);
}
```

Αρχείο: pitHndlr.asm

```
;Interrupt handler for programmable interval timer(pit)
[BITS 32]

GLOBAL _pitHndlr
```

```
EXTERN contexSwitch
EXTERN _PicEoi

%define PIT_IRQ    0

_pitHndlr:
        call contexSwitch

       ;code here to update the pit (if need)


        ;does not preserve registers
        push PIT_IRQ
        call _PicEoi
        add esp, 0x4

        ;push eax
        ;push edx

;        mov al, 0x20
;        mov dx, 0xA0
;        out dx, al
;        mov dx, 0x20
;        out dx, al

;        pop edx
;        pop eax

        ;mov eax, [esp + 16]
        ;call print
        iret
```

```
CharTable db '0', '1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'

  print:
        mov edx, 0xb8000
        mov ecx, 32

        next:

        sub ecx, 4
        mov ebx, eax
        shr ebx, cl
        and ebx, 0x0F
        mov ebx, [CharTable + ebx]
        mov [edx], bx
        inc edx
        mov byte [edx], 0x0F
        inc edx

        cmp ecx, 0
        je f

        jmp next


        f: jmp f
```

Αρχείο: pit_interrupt.asm

```
[BITS 32]

section .text
```

```
GLOBAL _pit_interrupt_handler
EXTERN _selectProcces
EXTERN _showStack
;EXTERN _clearScreen
;EXTERN _ab



align 4
_pit_interrupt_handler:



        pushad
;push eax
        ;CLI
        std
        call _selectProcces
        ;call _showStack
;L:jmp L
;jmp 0x08:_showStack ;0xd98

;pop eax
        popad
        ;sti
        iret
```

Αρχείο: ProcCondition.c

```
#include "procCondition.h"
#include "proccessStruct.h"
#include "destroyProccess.h"
#include "stdiok.h"
```

```
extern pcbNode *currentProcces, *readyProccesQueue, *blockedProccesQueue; /**??include file*/

size_t cond_inc = 1;

void init_condition(cond_t *cond)
{
   cond->code = cond_inc++;
   //cond->proc = NULL;
}

void cond_add_proc(cond_t *cond, pcbNode *curr)
{
   //cond->proc = curr;
}

void cond_destroy(void)
{

}

bool cond_signal(cond_t *cond)
{
   pcbNode *curr = blockedProccesQueue;


   while(curr != NULL)
   {

      if(curr->msgQueue.cond_emtpyQueue.code == cond->code )
      {
         unblockProccess(curr);
```

```
            return true;
        }

        curr = curr->next;
    }

    return false;
}
```

Αρχείο: procScheduler.c

```
#include "procScheduler.h"
#include "kernelHeap.h"
#include "stdio.h"
#include "registers.h"
#include "io.h"
```

```
#include "stdiok.h"



pcbNode *readyProccesQueue, *currentProcces, *readyProccesQueueTail,
*blockedProccesQueue;
extern pcbNode *ioBlockedProccesQueue;
dword newPid = 1;

pcbNode *getPcbByPid(dword);
void setCurrEsp0(dword esp0);



dword createNewPid(void)
{
   return newPid++;
}

void initProc(void)
{
   readyProccesQueue = readyProccesQueueTail = currentProcces = blockedProccesQueue =
ioBlockedProccesQueue = NULL;
}

/*this function need cli*/
/*#define KERNEL_SPACE_EFALGS 0x216
#define KERNEL_CODE_SEGMENT 0x08
#define USER_CODE_SEGMENT 0x23
dword newProcces(address entryPoint)
{
   childProccesNode *newChildId = NULL;
   pcbNode *tempCurr = currentProcces; ///holds currentProcces's value
   pcbNode *newProc = kmalloc(sizeof(pcbNode));
   register32 userEsp;
```

```
   if(newProc == NULL)
      return 1;

   currentProcces = newProc;

   if(readyProccesQueue == NULL)
      readyProccesQueue = readyProccesQueueTail = newProc;
   else
   {
      readyProccesQueueTail->next = newProc;
      readyProccesQueueTail = readyProccesQueueTail->next;
   }
   readyProccesQueueTail->next = NULL;

   newProc->pageCat.count = 1;
   if((newProc->pageCat.catalog = createProccessCatalog()) == NULL)
      return 1;

   if((newProc->ebp = (register32) ScAllocatePhysicalPages(1)) == 0)
      return 1;

   if((userEsp = (register32) ScAllocatePhysicalPages(1)) == 0)
      return 1;

   userEsp += (register32) PAGE_FRAME_SIZE - 4;

   newProc->ebp += PAGE_FRAME_SIZE - 4; ///first byte of the highest dword on stack    //this
must change if stack size change!!!
   newProc->esp = newProc->ebp - 52 + 4;
   (*((dword*)newProc->ebp)) = 0x1B;      //user data segment
   (*((dword*)(newProc->ebp - 4 ))) = userEsp;
   (*((dword*)(newProc->ebp - 8) )) = KERNEL_SPACE_EFALGS;
```

```
   (*((dword*)(newProc->ebp - 12) )) = USER_CODE_SEGMENT;
   (*((dword*)(newProc->ebp - 16) )) = (dword) entryPoint;
   (*((dword*)(newProc->ebp - 36) )) = (dword) newProc->ebp - 8;   ///ESP** THIS VALUE PROBABLY IGNORED!!
   (*((dword*)(newProc->ebp - 40) )) = (dword) newProc->ebp;     ///EBP

//printf("%x", (dword)(*((dword*)newProc->ebp - 4)));
   newProc->esp0 = newProc->ebp;

   newProc->pid = createNewPid();
   newProc->state = ready;

   if(tempCurr != NULL)
   {
      newProc->parentPid = tempCurr->pid;
      newProc->parentPcb = tempCurr;
   }
   else
   {
      newProc->parentPid = 0;
      newProc->parentPcb = NULL;
   }

   if(tempCurr != NULL)
   {
      if(tempCurr->childPidListHead == NULL)
      {
         tempCurr->childPidListHead = kmalloc(sizeof(childProccesNode));   ///Not checking if kmallod fails
         tempCurr->childPidListHead->next = NULL;
      }
      else
      {
```

```
            newChildId = kmalloc(sizeof(childProccesNode));
            newChildId->next = tempCurr->childPidListHead;
            tempCurr->childPidListHead = newChildId;
        }

        tempCurr->childPidListHead->hasReturn = false;
        tempCurr->childPidListHead->pid = newProc->pid;
    }

    newProc->childPidListHead = NULL;

    //printf("procces created!!!: %x\n", (dword)entryPoint);

    currentProcces = tempCurr;

    return 0;
} */

void selectProcces(void)
{
    dword newEbp, newEsp;/*DWORD*/
    address ebp;

    if(readyProccesQueue == NULL)
    {
        __asm volatile("sti;"
                "hlt;");
    }

    if(readyProccesQueue != NULL)
    {
        GET_EBP(ebp);
```

```
    if(currentProcces != NULL)
    {
       currentProcces->ebp = (dword) (*(dword*)(ebp));
       currentProcces->esp = (dword) ebp + 8;
    }

    if(readyProccesQueue->next != NULL)
    {
       currentProcces = readyProccesQueue->next;
       readyProccesQueueTail->next = readyProccesQueue;
       readyProccesQueueTail = readyProccesQueueTail->next;
       readyProccesQueueTail->next = NULL;
       readyProccesQueue = currentProcces;
    }
    else
       currentProcces = readyProccesQueue;

    setCurrEsp0(currentProcces->esp0);

    newEbp = currentProcces->ebp;
    newEsp = currentProcces->esp;

    SET_DS(0x1B);
    SET_ES(0x1B);
    SET_FS(0x1B);
    SET_GS(0x1B);

    SET_ESP((dword) (newEsp ) );
    SET_EBP((dword) newEbp );
}
```

```
   OUTB(0x20, 0xA0);
   OUTB(0x20, 0x20);

   __asm volatile("popa;");
   __asm volatile("iret;");
}

pcbNode *getPcbByPid(dword pid)
{
   pcbNode *curr = readyProccesQueue;

   while(curr != NULL && curr->pid != pid)
      curr = curr->next;

   if(curr == NULL)
   {
      curr = blockedProccesQueue;
      while(curr != NULL)
         curr = curr->next;
   }

   return curr;
}
/*
void exitPorcces(int ret)///exitProcces is better
{
   childProccesNode *temp = NULL;
   pcbNode *childPcb = NULL;

   if(currentProcces != NULL && currentProcces == readyProccesQueue)
   {
      destroyProccesCatalog();
```

```
///rearange pid

if(readyProccesQueue->next != NULL)
{
   readyProccesQueue = readyProccesQueue->next;
}
else
   readyProccesQueue = NULL;

if( currentProcces->parentPcb->state == blocked )
{
  if(blockedProccesQueue->pid == currentProcces->parentPid)
  {
     blockedProccesQueue = blockedProccesQueue->next;
  }
  else
  {
     pcbNode *temp = blockedProccesQueue;
     while(temp->next->pid != currentProcces->parentPid && temp != NULL)
         temp = temp->next;

     temp->next = temp->next->next;
  }

  if(readyProccesQueue == NULL)
  {
     readyProccesQueue = currentProcces->parentPcb;
     readyProccesQueue->next = NULL;
  }
  else
  {
```

```
            readyProccesQueueTail->next = currentProcces->parentPcb;
            readyProccesQueueTail = readyProccesQueueTail->next;
            readyProccesQueueTail->next = NULL;
          }

          currentProcces->parentPcb->state = ready;
        }

        if(currentProcces->parentPcb != NULL)
        {
          temp = currentProcces->parentPcb->childPidListHead;///Not checking if childPidHead is
null

          if(temp == NULL)       //parent procces inherit current's procces childs
            currentProcces->parentPcb->childPidListHead = currentProcces->childPidListHead;

          while(temp != NULL)
          {
            if(temp->pid == currentProcces->pid)
            {
              temp->hasReturn = true;
              temp->retVal = ret;
            }

            if(temp->next == NULL)
              temp->next = currentProcces->childPidListHead; //parent procces inherit current's
procces childs

            temp = temp->next;
          }
        }

        temp = currentProcces->childPidListHead;
```

```
        while(temp != NULL)
        {
            if( (childPcb = getPcbByPid(temp->pid)) != NULL )
            {
                childPcb->parentPid = currentProcces->parentPid;
                childPcb->parentPcb = currentProcces->parentPcb;
            }

            temp = temp->next;
        }

        kfree(currentProcces);
        currentProcces = NULL;


        __asm volatile("int $32;");//selectProcces();
    }
}*/

int wait2(int *status)
{
    childProccesNode *curr;

    if(currentProcces != NULL)
    {
        currentProcces->state = blocked;

        if(currentProcces->childPidListHead == NULL)
            return -1;

        curr = currentProcces->childPidListHead;
        while(curr != NULL)
```

```
      {
         if(curr->hasReturn == true)
         {
            /*free list node */
            *status = curr->retVal;
            return curr->pid;
         }
         curr = curr->next;
      }

      readyProccesQueue = readyProccesQueue->next;
      if(blockedProccesQueue == NULL)
      {
         //kmalloc(sizeof(pcbNode));
         blockedProccesQueue = currentProcces;
         currentProcces->next = NULL;

      }
      else
      {
         currentProcces->next = blockedProccesQueue;
         blockedProccesQueue = currentProcces;
      }

      __asm volatile("int $32;");//selectProcces();
      ///return pid
   }

   return -1;
}

#define TSS_ESP0_PH_ADD    (dword*)0x3A     /*??Kernel stack address of current procces must
always stored in that address*/
```

```
void setCurrEsp0(dword esp0)
{
   *TSS_ESP0_PH_ADD = esp0;
}
```

Αρχείο: protected.asm

```
[BITS 16]
[ORG 0x7C00]
;KERNEL_SIZE is pre-defined macro

   CLI
```

```
    CALL checkA20
    CMP AX, 0x00
;(If yes)
    JE copyBoot
;Use bios to enable a20 and check again
    MOV ax, 0x2401
    INT 0x15
    CALL checkA20
    CMP AX, 0x00
;(If yes)
    JE copyBoot
;fast a20 and keyboard need here

;if none of the methods word print message and hlt
    MOV SI, StringA20dis
    CALL PrintString
    HLT

copyBoot:
;copy the bootloader at 0x100000
    PUSH DS
;CX is store the number of word to copy
    MOV CX, 256;size of the bootloader

    MOV AX, 0xFFFF
    MOV ES, AX
    XOR AX, AX
    MOV DS, AX

    MOV SI, $$
    MOV DI, 0x10
```

```
   REP MOVSW
   POP DS
   JMP 0xFFFF:(loadKernel - $$ + 0x10)

loadKernel:
;Load sectors starting from 0x500 address (only first 72 because bochs's bug)
   STI
         XOR AX, AX
         MOV ES, AX

         MOV AH, 0x02
         MOV AL, 72
         MOV CX, 0x02
         MOV DL, 0x00
         MOV DH, 0x00
         MOV BX, 0x500
         INT 0x13

;Check if succed
         JC  loaderror

;Load sectors remain

         XOR AX, AX
         MOV ES, AX

         MOV AH, 0x02
         MOV AL, KERNEL_SIZE
         SUB AL, 72
         INC AL
```

```
        MOV CL, 0x02
        MOV CH, 0x01
        MOV DL, 0x00
        MOV DH, 0x00
        MOV BX, 0x9500
        INT 0x13

;Check if succed
        JC  loaderror

;load memory map
;LOAD MEMORY MAP AT 0x100204( ES = 0xFFFF: DI = 0x214)
        ;7e00
        XOR EBP, EBP
        MOV AX, 0xFFFF
        MOV ES, AX
        MOV AX, 0x214
        MOV DI, AX
        CLC
        XOR EBX, EBX
        MOV EDX, 0x534D4150 ;MAGIC NUMBER

nexEnt: MOV EAX, 0xE820
        MOV ECX, 0x18
        INT 0x15
        JC setEntryCount
        OR EBX, EBX
        JZ setEntryCount
        INC EBP
        ADD DI, 0X18
        JMP nexEnt
```

```
;set number of entries
setEntryCount:
        MOV  EAX, EBP
        MOV  [ES:0x210], EAX

;copy gdt at 0x00:0x00
CLI
PUSH DS

MOV CX, 48
MOV AX, 0x00
MOV ES, AX
MOV AX, 0xFFFF
MOV DS, AX
MOV SI, (0x10 + gdt - $$)
MOV DI, 0x00
REP MOVSB
POP DS

;create gdtr at 00:48 with

MOV WORD [ES:48], 47    ;size of gdt - 1
MOV DWORD [ES:50], 0x00 ;Address of gdt

;Set video mode
XOR AH, AH
MOV AL, 0x12
INT 0x10

;Enable protected mode and jump to kernel
CLI
```

```
LGDT [ES:48]
MOV EAX, CR0
OR AL, 1
MOV CR0, EAX

JMP 0x08:0x500 ;far jump to main

a20enabled:
MOV SI, StringA20ena
CALL PrintString
HLT



loaderror:
CLI
MOV SI, LoadErrorString
;CALL PrintString
HLT

;-------------------------------------------------
;Check A20 line
;If A20 is enabled return 0 on AX otherwise non zero value
checkA20:

;put a random number at 1M + 500 address
XOR EAX, EAX
MOV AX, 0xFFFF
MOV ES, AX
MOV AX, 0x4C41 ;RANDOM NUMBER
MOV [ES:0x510], EAX
```

```
;put 0xFFFF number 500 address
MOV AX, 0x0000
MOV ES, AX
MOV AX, 0xFFFF
MOV [ES:0x500], AX

;Check if 1M + 500 address keep his value
MOV AX, 0xFFFF
MOV ES, AX
MOV AX, [ES:0x510]
XOR AX, 0x4C41

RET
;-------------------------------------------------
;-------------------------------------------------
PrintCharacter: ;Procedure to print character on screen
                ;Assume that ASCII value is in register AL
PUSH BX
MOV AH, 0x0E   ;Tell BIOS that we need to print one charater on screen.
MOV BH, 0x00   ;Page no.
MOV BL, 0x07   ;Text attribute 0x07 is lightgrey font on black background
POP BX

INT 0x10       ;Call video interrupt
RET

;-------------------------------------------------
PrintString:    ;Procedure to print string on screen
        ;Assume that string starting pointer is in register SI

next_character: ;Lable to fetch next character from string
        MOV AL, [SI]    ;Get a byte from string and store in AL register
```

```
        INC SI              ;Increment SI pointer
        OR AL, AL           ;Check if value in AL is zero (end of string)
        JZ exit_function ;If end then return
        CALL PrintCharacter ;Else print the character which is in AL register
        JMP next_character      ;Fetch next character from string
exit_function:    ;End label
        RET

;-----------------------------------------------

StringA20dis DB "A20 is disabled", 0
StringA20ena DB "A20 is enable", 0
LoadErrorString DB "Looad error", 0


gdt:
dq 0x0 ;first entry of gdt 0
;second entry is the code segment:
dw 0xFFFF ;Second entry limit
dw 0x0   ;base address
db 0x0    ;base address
db 0x9A     ;type
db 0xCF
db 0x00 ;end of second entry
;third entry for all data: stack data es...:
dw 0xFFFF
dw 0x0   ;base address
db 0x0    ;base address
db 0x92
db 0xCF
db 0x00 ;end of third entry code segment
;user data segment
```

```
dw 0xFFFF
dw 0x0
db 0x0
db 0xF2
db 0xCF
db 0x00
;user code segment
dw 0xFFFF
dw 0x0
db 0x0
db 0xFA
db 0xDF
db 0x00
;tss segment  tss located at 00:54(after gdtr)
dw 0x67    ;tss limit
dw 0x36    ;address of tss
db 0x00
db 0xE9
db 0x00
db 0x00

TIMES 510 - ($ - $$) DB 0
DW 0xAA55
```

Αρχείο: shedule.c

```
#include "proccessStruct.h"
#include "type.h"
#include "procScheduler.h"
#include "stdiok.h"
```

```
extern pcbNode *readyProccesQueue, *readyProccesQueueTail, *currentProcces;
extern emptyStack;

static void  plcElmntToEnd(pcbNode**, pcbNode**);

void shedule(void)
{

   if(readyProccesQueue == NULL)
   {
      /*a hlt procces must be created here*/
      //printk("ready procces queue is empty!");
      currentProcces = NULL;
      setCurrEsp0(emptyStack+32);
      return;
   }

   if(readyProccesQueue->next != NULL)
   {
      plcElmntToEnd(&readyProccesQueue, &readyProccesQueueTail);
      currentProcces = readyProccesQueue;


   }
   else /*if readyProccesQueue->next == NULL*/
   {
      currentProcces = readyProccesQueue; /*Continue execution of the same proccess if there
are no other ready procceses*/
   }

   //printk("c: %x\n", (dword)(*(dword*)(currentProcces->esp0 - 8)) );
   //while(1);

   setCurrEsp0(currentProcces->esp0);
```

```
}

void saveKrnlStck(dword esp)
{
   if(currentProcces != NULL)
   {
      currentProcces->esp = esp;
   }
}

dword loadKrnlStck(void)
{
   if(currentProcces != NULL)
      return currentProcces->esp;
   return (dword) 0;
}

static void plcElmntToEnd(pcbNode **startQueue, pcbNode **endQueue)
{
   (*endQueue)->next = *startQueue;
   *startQueue = (*startQueue)->next;
   *endQueue = (*endQueue)->next;
   (*endQueue)->next = NULL;

}
```

Αρχείο: stdio.c

```
#include "stdio.h"
#include "type.h"
#include "kernelHeap.h"
#include "sysCallLib.h"
```

```
heapNode *memListHead;

static void* createReservedNode(heapNode *curr, size_t allocSize);
static heapNode* findFreeNode(size_t allocSize);
heapNode* findReservedRegion(void* ptr, heapNode** prevFree);
heapNode* mergeIfNeighbour(heapNode *prev, heapNode *curr);

size_t temp_vidIndx;
byte emptyDraw[80*25];

void stdInit(void)
{
   temp_vidIndx = 0;
   memset(emptyDraw, 0, 80*25);  ///memset alternative
}

static size_t vid_inc(size_t *t)
{
   size_t res = *t;

   if(*t < 1279)
      (*t)++;
   else{
      *t = 0;
      draw(emptyDraw, 0, 80*25);
   }

   return res;
}

int printf(char *ch, ...)
```

```
{
   va_list arg;
   dword val, i, j = 28;
   char symbolTable[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
   size_t pad;

   va_start(arg, ch);

   if(ch == NULL)
      return -1;

   while(*ch != '\0')
   {
      switch(*ch){
         case '%':
            ch++;
            switch( *ch ){
               case '%':
                  //ScPrintChar(*ch, 0x0F);
                  draw((byte*) ch, vid_inc(&temp_vidIndx), 1);
                  break;
               case 'x':
                  val =(dword) va_arg(arg, dword);
                  for(i =(dword) 0xF0000000; i >= (dword)0xF; i >>= 4, j-=4 )
                     draw((byte*)&symbolTable[ (val & i) >> j ], vid_inc(&temp_vidIndx),
1);//ScPrintChar(symbolTable[ (val & i) >> j ], 0x0F);
                  break;
               case 'c':
                  val =(byte) va_arg(arg, byte);
                  draw((byte*)&val, vid_inc(&temp_vidIndx), 1);  //ScPrintChar(val, 0x0F);
                  break;

            }
```

```
            break;
            case '\n':

               if(temp_vidIndx > 24*80){
                  draw(emptyDraw, 0, 80*25);
                  temp_vidIndx = 0;
               }
               else{
                  pad = 80-(temp_vidIndx % 80);
                  draw(emptyDraw, temp_vidIndx, pad );
                  temp_vidIndx += pad;
               }

               break;

            default:
               draw((byte*)ch, vid_inc(&temp_vidIndx), 1);  //ScPrintChar(*ch, 0x0F);
               break;


         }

         ch++;
      }

      va_end(arg);

      return 0;
   }
```

```
void memcpy(byte *source, byte *dest, size_t count)
{
      while(count--)
         *dest++ = *source++;
}

void memset(void* buffer, int val, size_t count)
{
   char* p = buffer;

   while(count--)
      *p++ = (unsigned char) val;
}

void initKernelHeap(void)
{
   memListHead = (heapNode*) KERNEL_HEAP_START_ADD;
   memListHead->next = NULL;
   memListHead->size = (size_t) (KERNEL_HEAP_END_ADD - KERNEL_HEAP_START_ADD -
sizeof(heapNode) );
   memListHead->isFreeSlot = true;

}

void *kmalloc(size_t allocSize)
{
   heapNode *curr = NULL;

   if((curr = findFreeNode(allocSize)) == NULL)
      return NULL;

   return createReservedNode(curr, allocSize);
}
```

```
static heapNode* findFreeNode(size_t allocSize)
{
   heapNode *curr = memListHead;

   while(curr != NULL)
   {
      if( (curr->size >= allocSize + sizeof(heapNode) ) && (curr->isFreeSlot == true) )
         return curr;

      curr = curr->next;
   }

   return NULL;
}

static void* createReservedNode(heapNode *curr, size_t allocSize)
{
   heapNode *newNode;

   if(curr->size - (allocSize + sizeof(heapNode)) > sizeof(heapNode))
   {
      newNode = (heapNode*) ((size_t) curr + allocSize + sizeof(heapNode));
      newNode->next = curr->next;
      curr->next = newNode;

      newNode->size = curr->size - allocSize - sizeof(heapNode);
      curr->size = allocSize;
      curr->isFreeSlot = false;
      newNode->isFreeSlot = true;

      return (void*)((size_t) curr + sizeof(heapNode));
```

```
    }
    else if(curr->size >= allocSize + sizeof(heapNode))
    {
        curr->isFreeSlot = false;
        return (void*)((size_t) curr + sizeof(heapNode));
    }

    return NULL;
}
void kfree(void *ptr)
{
    heapNode *curr = NULL, *prevFreeNode = NULL;

    if( (curr = findReservedRegion(ptr, &prevFreeNode)) == NULL)
        return;

    curr->isFreeSlot = true;

    curr = mergeIfNeighbour(prevFreeNode, curr);
    curr = mergeIfNeighbour(curr, curr->next);

    return;
}

heapNode* mergeIfNeighbour(heapNode *prev, heapNode *curr)
{
    if(prev == NULL)return curr;
    if(curr == NULL)return prev;

    if(prev->next == curr)
    {
        prev->size += curr->size + sizeof(heapNode);
```

```
        prev->next = curr->next;

        return prev;
    }
    else
        return curr;
}

heapNode* findReservedRegion(void* ptr, heapNode** prevFree)
{
    heapNode *curr = memListHead;

    *prevFree = NULL;
    while(curr != NULL)
    {
        if( (ptr == (void*) ((size_t) curr + sizeof(heapNode)) && (curr->isFreeSlot == false)) )
            return curr;

        if(curr->isFreeSlot == true)
            *prevFree = curr;

        curr = curr->next;
    }
    return NULL;
}
```

Αρχείο: stdiok.c

```
#include "stdiok.h"
#include "vga.h"
#include "stdarg.h"

int printk(char *ch, ...)
```

```
{
   va_list arg;
   dword val, i, j = 28;
   char symbolTable[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};

   va_start(arg, ch);

   if(ch == NULL)
      return -1;

   while(*ch != '\0')
   {
      switch(*ch){
         case '%':
            ch++;
            switch( *ch ){
               case '%':
                  printChar(*ch, 0x0F);
                  break;
               case 'x':
                  val =(dword) va_arg(arg, dword);
                  for(i =(dword) 0xF0000000; i >= (dword)0xF; i >>= 4, j-=4 )
                     printChar(symbolTable[ (val & i) >> j ], 0x0F);
                  break;
               case 's':
                  val = (dword)  va_arg(arg, dword);
                  printk( (char*) val );
                  break;
               case 'c':
                  val =(byte) va_arg(arg, byte);
                  printChar((char)val, 0x0F);
                  break;
```

```
            }

        break;
        default:
            printChar(*ch, 0x0F);
            break;


    }

    ch++;
  }

  va_end(arg);

  return 0;
}
```

Αρχείο: string.c

```
#include "string.h"

size_t strlen(const char* str)
{
  size_t i = 0;
```

```
    while(str[i] != '\0')
        i++;

    return i;
}

char *strncpy(char *destination, const char *source, size_t num)
{
    size_t i = 0;

    while(i < num && source[i] != '\0')
    {
        destination[i] = source[i];
        i++;
    }

    while(i < num)
        destination[i++] = '\0';

    return destination;
}

char *strncat(char *destination, const char *source, size_t num)
{
    size_t i = 0, j = strlen(destination);

    while(i < num  && source[i] != '\0')
        destination[j++] = source[i++];

    destination[j] = '\0';
    return destination;
```

```
}

char *strcpy(char *destination, const char *source)
{
   size_t i = 0;

   do
   {
      destination[i] = source[i];
      i++;
   }while( source[i] != '\0');

   return destination;
}

char *strcat(char *destination, const char *source)
{
   size_t i = 0, j = strlen(destination);

   do
   {
      destination[j++] = source[i++];
   }while(source[i] != '\0');

   return destination;
}

int strncmp(const char *str1, const char *str2, size_t num)
{
   size_t i = 0;

   while(i < num && (str1[i] != '\0' || str2[i] != '\0') )
```

```
   {
      if(str1[i] < str2[i])
         return -1;
      else if(str1[i] > str2[i])
         return 1;

      i++;
   }

   return 0;
}

int strcmp(const char *str1, const char *str2)
{
   size_t i = 0;

   while( str1[i] == str2[i]  )
   {
      if(str1[i] == '\0')
         return 0;

      i++;
   }

   if(str1[i] < str2[i])
      return -1;
   else
      return 1;

}

size_t strcspn(const char *str1, const char *str2)
```

```
{
   size_t cnt, i;

   for(cnt = 0;  cnt < strlen(str1);  cnt++ )
   {
      i = 0;
      while( i++ < strlen(str2) )
         if(str1[cnt] == str2[i] )
            return cnt;
   }

   return cnt;
}

/**This function is not part of the c library*/
char *inttostr(size_t val, size_t base, char *destination)
{
   const int ascii_zero_char_offset = 48;
   char digit = val % base;

   if(base > 10)
      return NULL;

   if(val / base != 0)
      inttostr(val / base, base, destination);

   digit += ascii_zero_char_offset;
   strncat(destination, &digit, 1);

   return destination;
}
```

```
/**This function is not part of the c library*/
bool isLetter(const char ch)
{
   const char ascii_A_offset = 65, ascii_Z_offset = 90, ascii_a_offset = 97, ascii_z_offset = 122;

   return ( (ch >= ascii_A_offset && ch <= ascii_Z_offset) || (ch >= ascii_a_offset && ch <= ascii_z_offset))? true : false;
}

/**This function is not part of the c library*/
size_t copyBytes(byte *destination, const byte *source, size_t num)
{
   size_t i;

   for(i = 0;  i < num; i++)
      destination[i] = source[i];

   return num;
}
```

Αρχείο: sysCallHandler.asm

```
global _sys_call_hndlr
extern _sysCallTable

%define SYS_CALLS_CNT    19   ;Number of syscalls
```

```
CharTable db '0', '1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'

write:
        mov ecx, 0xb8000
        mov eax, ebx

        shr ebx, 28
        mov edx, [CharTable + ebx]
        mov [ecx], edx
        inc ecx
        mov edx, 0x0F
        mov [ecx], edx
        inc ecx

        mov ebx, eax
        shr ebx, 24
        and ebx, 0xF
        mov edx, [CharTable + ebx]
        mov [ecx], edx
        inc ecx
        mov edx, 0x0F
        mov [ecx], edx
        inc ecx

        mov ebx, eax
        shr ebx, 20
        and ebx, 0xF
        mov edx, [CharTable + ebx]
        mov [ecx], edx
        inc ecx
        mov edx, 0x0F
        mov [ecx], edx
```

```
inc ecx

mov ebx, eax
shr ebx, 16
and ebx, 0xF
mov edx, [CharTable + ebx]
mov [ecx], edx
inc ecx
mov edx, 0x0F
mov [ecx], edx
inc ecx

mov ebx, eax
shr ebx, 12
and ebx, 0xF
mov edx, [CharTable + ebx]
mov [ecx], edx
inc ecx
mov edx, 0x0F
mov [ecx], edx
inc ecx

mov ebx, eax
shr ebx, 8
and ebx, 0xF
mov edx, [CharTable + ebx]
mov [ecx], edx
inc ecx
mov edx, 0x0F
mov [ecx], edx
inc ecx
```

```
        mov ebx, eax
        shr ebx, 4
        and ebx, 0xF
        mov edx, [CharTable + ebx]
        mov [ecx], edx
        inc ecx
        mov edx, 0x0F
        mov [ecx], edx
        inc ecx

        mov ebx, eax
        shr ebx, 0
        and ebx, 0xF
        mov edx, [CharTable + ebx]
        mov [ecx], edx
        inc ecx
        mov edx, 0x0F
        mov [ecx], edx
        inc ecx

        ret

_sys_call_hndlr:
        cmp eax, SYS_CALLS_CNT
        jae  sys_call_not_sup
        ;mov ebx, eax
        ;call write

        push ebp    ;sixth param
        push edi
        push esi
        push edx
```

```
        push ecx
        push ebx    ;first param
        call [_sysCallTable + eax * 4]
        pop ebx
        pop ecx
        pop edx
        pop esi
        pop edi
        pop ebp

        iret


sys_call_not_sup:   mov eax, -1
        iret
```

Αρχείο: sysCallLib.c

```
#include "sysCallLib.h"
#include "registers.h"

makeSysCall2(void, ScPrintChar, char, charToPrint, byte, color)
```

makeSysCall1(void*, allocatePhysicalPages, size_t, length)

makeSysCall1(int, freePhysicalPages, void*, ptr)

makeSysCall1(dword, createProcces, const char *, entry)

makeSysCall1(void, exitProc, int, ret)

makeSysCall1(int, waitProc, int*, status)

makeSysCall3(dword, readFile, dword, fileHndlr, size_t, count, byte*, buffer)

makeSysCall4(dword, writeFile, dword, fileHndlr, size_t, count, byte*, buffer, int, flags)

makeSysCall2(dword, openFile, const char *, path, const char *, rw)

makeSysCall3(dword, seekFile, dword, fileHndlr, long int, offset, int, origin)

makeSysCall1(void, closeFile, dword, filehndlr)

makeSysCall1(dword, getMessage, void*, msg)

makeSysCall1(dword, setProcProperties, dword, procDescr)

makeSysCall1(dword, setActiveProc, dword, procHndl)

makeSysCall3(void, draw, byte*, buffer, size_t, offset, size_t, count)

makeSysCall1(dword, fileExists, const char *, path)

makeSysCall4(dword, getFileAttr, const char *, path, const dword, fromFile, size_t, countFiles, fileAttr *, fileAtt)

Αρχείο: test.asm

```
global _sys_call_hndlr
extern _sysCallTable

%define SYS_CALLS_CNT    6   ;Number of syscalls
```

```
CharTable db '0', '1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'

write:
        mov ecx, 0xb8000
        mov eax, ebx

        shr ebx, 28
        mov edx, [CharTable + ebx]
        mov [ecx], edx
        inc ecx
        mov edx, 0x0F
        mov [ecx], edx
        inc ecx

        mov ebx, eax
        shr ebx, 24
        and ebx, 0xF
        mov edx, [CharTable + ebx]
        mov [ecx], edx
        inc ecx
        mov edx, 0x0F
        mov [ecx], edx
        inc ecx

        mov ebx, eax
        shr ebx, 20
        and ebx, 0xF
        mov edx, [CharTable + ebx]
        mov [ecx], edx
        inc ecx
        mov edx, 0x0F
        mov [ecx], edx
```

```
inc ecx

mov ebx, eax
shr ebx, 16
and ebx, 0xF
mov edx, [CharTable + ebx]
mov [ecx], edx
inc ecx
mov edx, 0x0F
mov [ecx], edx
inc ecx

mov ebx, eax
shr ebx, 12
and ebx, 0xF
mov edx, [CharTable + ebx]
mov [ecx], edx
inc ecx
mov edx, 0x0F
mov [ecx], edx
inc ecx

mov ebx, eax
shr ebx, 8
and ebx, 0xF
mov edx, [CharTable + ebx]
mov [ecx], edx
inc ecx
mov edx, 0x0F
mov [ecx], edx
inc ecx
```

```
        mov ebx, eax
        shr ebx, 4
        and ebx, 0xF
        mov edx, [CharTable + ebx]
        mov [ecx], edx
        inc ecx
        mov edx, 0x0F
        mov [ecx], edx
        inc ecx

        mov ebx, eax
        shr ebx, 0
        and ebx, 0xF
        mov edx, [CharTable + ebx]
        mov [ecx], edx
        inc ecx
        mov edx, 0x0F
        mov [ecx], edx
        inc ecx

        ret

_sys_call_hndlr:
        cmp eax, SYS_CALLS_CNT
        jae  sys_call_not_sup
        ;mov ebx, eax
        ;call write

        push ebp    ;sixth param
        push edi
        push esi
        push edx
```

```
        push ecx
        push ebx    ;first param
        call [_sysCallTable + eax * 4]
        pop ebx
        pop ecx
        pop edx
        pop esi
        pop edi
        pop ebp

        iret


sys_call_not_sup:   mov eax, -1
        iret
```

Αρχείο: vfs.c

```
#include "vfs.h"
#include "stdiok.h"
#include "ide.h" /*this is temp*/
#include "device.h"
#include "volume.h"
```

```
#include "string.h"
#include "file.h"
#include "fileSystem.h"

#define INPUT_C 0x1
#define OUTPUT_C 0x2

#define INVALID_PATH    0x00
#define IS_DEVICE       0x01
#define IS_VOLUME       0x02
#define IS_FILE         0x03



extern pcbNode *currentProcces;

static byte getPathType(const char *path);
static int searchDevice(const char *path);
static int searchVolume(const char *path);
static size_t getHndlFromPath(const char *path);/**temp function*/
static void splitPath(const char *path, char *vol, char *file);

dword vfread(const char *path, dword *fsIndx, size_t offset, size_t count, void *buffer)
{
   char vol[4], file[PATH_MAX_LENGTH];
   dword tfsIndx;
   int index;
   size_t readcount;

   switch(getPathType(path)){
   case INVALID_PATH:
      printk("invalid1 path\n");
```

```
        return 0;
        break;
    case IS_DEVICE:
        index = searchDevice(path);
        if(index == -1)
        {
            printk("is device with index %x\n", (dword) index);
            return 0;
        }
        if(devices[index].op.read != NULL)
        {
          readcount = devices[index].op.read(offset, count, buffer);/*get offset to read() first param
from file[i].pos*/
          //updateFileCursor(path, readcount);
          return readcount;
        }
        break;
    case IS_VOLUME:
        readcount = VolumeRead(path, offset, count, buffer);
        return readcount;
    case IS_FILE:
        splitPath(path, vol, file);
        index = (int) getFsIndx(vol);

        if(NULL != fsIndx)
            tfsIndx = *fsIndx;
        else
            tfsIndx = getFsIndFromPath(path);

        if( tfsIndx  != PATH_NOT_FOUND )
            return devices[index].op.read( tfsIndx, count, buffer);

        break;
```

```
    }

    return 0;
}

dword vfwrite(const char *path, size_t offset, size_t count, void *buffer, int flags)
{
    char vol[4], file[PATH_MAX_LENGTH];
    dword fsIndx;
    int index;
    size_t writecount;

    //printk("path is:\n");
    switch(getPathType(path)){
    case INVALID_PATH:
        printk("invalid2 path\n");
        return 0;
        break;
    case IS_DEVICE:
        index = searchDevice(path);
        if(index == -1)
        {
            printk("is device with index %x\n", (dword) index);
            return 0;
        }
        if(devices[index].op.write != NULL)
        {
            //printk("\ndev write...\n");
            writecount = devices[index].op.write(offset, count, buffer, flags);
            //printk("\ndev write after ...\n"); while(1);
            //updateFileCursor(path, writecount);
            printk("buffer[10] is: %x count is %x", (dword)((byte*)buffer)[10], (dword) count);
```

```
            printk("device write return %x\n", writecount);
            return writecount;
        }
        break;
    case IS_VOLUME:
        //printk("is volume\n");
        //testf();
        writecount = VolumeWrite(path, offset, count, buffer);
        break;
    case IS_FILE:
        splitPath(path, vol, file);
        index = (int) getFsIndx(vol);

        if( (fsIndx = getFsIndFromPath(path) ) != PATH_NOT_FOUND )
            return devices[index].op.write( fsIndx, count, buffer, flags);
        break;
    }

    return 0;
}

dword vfopen(const char *path, const char * rw, size_t * fsize)
{
    char vol[4], file[PATH_MAX_LENGTH];
    dword findx = 0xFFFFFFFF;
    int index;

    switch(getPathType(path))
    {
    case IS_DEVICE:
        if( (index = searchDevice(path)) == -1)
            return 0;    ///??return 0xFFFFFFFF
```

```
        if( devices[index].op.fopen == NULL)
            return 0;
        if(devices[index].op.fopen(path, rw) == 0)
            return 0;
        if(devices[index].op.getSize == NULL)
            return 0;
        *fsize = devices[index].op.getSize() * 512;
        break;
    case IS_VOLUME:
        if( (index = searchVolume(path)) == -1)
            return 0;
        *fsize = volumes[index].partitionSize * 512;
        break;
    case IS_FILE:
        splitPath(path, vol, file);
        *fsize = 0;
        findx = (int) getFsIndx(vol);
        index = devices[findx].op.fopen(file, rw);
        break;
    default:
        return 0;

    }

    return index;
}

dword vfcreate(const char *path)
{
    char vol[4], file[PATH_MAX_LENGTH];
    dword findx = 0xFFFFFFFF;
    int index;
```

```
    switch(getPathType(path))
    {
    case IS_DEVICE:
        return 0;
        break;
    case IS_VOLUME:
        return 0;
        break;
    case IS_FILE:
        splitPath(path, vol, file);
        findx = (int) getFsIndx(vol);
        if(devices[findx].op.fcreate(file)  == 1)
            index = Scfopen(path, "");
        break;
    default:
        return 0;

    }

    return index;
}

dword Scfopen(const char *path, const char * rw)
{
    size_t fsize;
    int index;

    index = vfopen(path, rw, &fsize);

    if(!index)
        return -1;
```

```
    return addFiletoPcb(path, fsize, index);
}

dword Scfcreate(const char *path)
{
    //return Scfopen("A:/OUT.RUN", "");

    size_t fsize;
    int index;

    index = vfcreate(path);

    if(!index)
        return -1;

    return addFiletoPcb(path, 0, index);
}

void vfclose(dword filehndlr)
{
    const char *path = getPathFromHandle(filehndlr);
    char vol[4], file[PATH_MAX_LENGTH];
    dword fsIndx = 0xFFFFFFFF;
    int index;

    if(path == NULL)
        return;

    switch(getPathType(path))
    {
    case IS_DEVICE:
```

```
      break;
   case IS_VOLUME:
      break;
   case IS_FILE:
      splitPath(path, vol, file);
      index = (int) getFsIndx(vol);

      if( (fsIndx = getFsIndFromPath(path) ) != PATH_NOT_FOUND )
         devices[index].op.fclose(fsIndx);
      break;
   }

}

void SccloseFile(dword filehndlr)
{
   vfclose(filehndlr);
   ///??remove file from pcb
}

dword ScReadFile(dword fileHndlr, size_t count, byte* buffer)
{
   const char *path = getPathFromHandle(fileHndlr);
   size_t file_cur = getPosFromHandle(fileHndlr);
   size_t bcnt = 0;

   if(path == NULL || file_cur == INVLD_FILE_CUR_POS)
      return 0;

   bcnt = vfread( path, NULL, file_cur, count, buffer);

   updateFileCursor(path, bcnt + file_cur);
```

```
//printf("read %x\n", bcnt);
   return (dword) bcnt;
}

dword ScWriteFile(dword fileHndlr, size_t count, byte* buffer, int flags)
{
   const char *path = getPathFromHandle(fileHndlr);
   size_t file_cur = getPosFromHandle(fileHndlr);
   size_t bcnt = 0;

   if(path == NULL || file_cur == INVLD_FILE_CUR_POS)
      return 0;

   bcnt = vfwrite( path, file_cur, count, buffer, flags);

   updateFileCursor(path, bcnt + file_cur);

   return (dword) bcnt;
}

dword ScseekFile(dword fileHndlr, long int offset, int origin )
{
   char vol[4], file[PATH_MAX_LENGTH];
   dword findx = 0XFFFFFFFF;
   size_t index;
   const char *path = getPathFromHandle(fileHndlr);

   if(path == NULL )
      return SC_FSSEK_FAILED;

   switch(getPathType(path))
   {
```

```
case IS_DEVICE:
   if( (index = searchDevice(path)) == -1)
      return SC_FSSEK_FAILED;
   if( devices[index].op.fseek != NULL)
      devices[index].op.fseek(0,0,origin);
   break;
//case IS_VOLUME:
   //if( (index = searchVolume(path)) == -1)

   //break;
case IS_FILE:
   splitPath(path, vol, file);
   index = (int) getFsIndx(vol);

   if( (findx = getFsIndFromPath(path) ) != PATH_NOT_FOUND )
      return (dword) devices[index].op.fseek(findx, (long64) offset, origin);

   break;
}



switch(origin)
{
case SEEK_SET:
   return updateFileCursor(path, offset);
   break;
case SEEK_CUR:
   return updateFileCursor(path, (offset + getPosFromHandle(fileHndlr)) );
   break;
case SEEK_END:
   return 0;
   break;
```

```
    }

    return 0;

}

dword vffileExists(const char *path, dword *attr, size_t *fsize)
{
    char vol[4], file[PATH_MAX_LENGTH];
    dword findx = 0xFFFFFFFF;
    int index = 0;

    switch(getPathType(path))
    {
    case IS_DEVICE:
        if( (index = searchDevice(path)) == -1)
            return 0;    ///??return 0xFFFFFFFF
        if( devices[index].op.fexists == NULL)
            return 0;  ///??return 0xFFFFFFFF

        return devices[index].op.fexists(path, NULL, NULL);
        break;

    case IS_VOLUME:
        if( (index = searchVolume(path)) == -1)
            return 0;
        else
            return 1;
        break;

    case IS_FILE:
        splitPath(path, vol, file);
```

```
      findx = (int) getFsIndx(vol); ///??return -1 same on fopen
      index = devices[findx].op.fexists(file, attr, fsize);

      break;
   }

   return index;
}

dword ScfileExists(const char *path, dword *attr, size_t *fsize)
{
   vffileExists(path, attr, fsize);
}

dword vGetFileAttr(const char* path, const dword fromFile, size_t countFiles,  fileAttr *fileAtt)
{
   char vol[4], file[PATH_MAX_LENGTH];
   dword findx = 0xFFFFFFFF;
   int count = 0;

   switch(getPathType(path))
   {
   case IS_DEVICE:
      return 0;
      break;
   case IS_VOLUME:
         return 0;
      break;
   case IS_FILE:
      splitPath(path, vol, file);
      findx = (int) getFsIndx(vol); ///??return -1 same on fopen
      count = devices[findx].op.list_dir(file, fromFile, countFiles, fileAtt);
```

```
      break;
   }

   return count;
}

dword ScGetFileAttr(const char* fpath, const dword fromFile, size_t countFiles,  fileAttr *fileAtt)
{
   return vGetFileAttr(fpath, fromFile, countFiles, fileAtt);
}

static byte getPathType(const char *path)
{

   if(path == NULL || path[0] == '\0' || strlen(path) > PATH_MAX_LENGTH)
      return INVALID_PATH;

   if(strncmp(path, STORAGE_DEV_STRING, strlen(STORAGE_DEV_STRING)) == 0)
      return IS_DEVICE;
   else if( isLetter(path[0])  && path[1]  == ':' && path[2] == '\0')
      return IS_VOLUME;
   else if( isLetter(path[0])  && path[1]  == ':' && path[2] != '\0' )
      return IS_FILE;
   else
      return INVALID_PATH;
}

static int searchDevice(const char *path)
{
   int i;
```

```
   for(i = 0;  i < MAX_DEV_SUPPORTED; i++)
   {
      if(devices[i].type == STORAGE)
      {
         if( strcmp(devices[i].pathId, path) == 0)
         {

            return i;
         }
      }
   }

   return -1;
}

static int searchVolume(const char *path)
{
   int i;

   for(i = 0;  i < MAX_VOLUMES_COUNT; i++)
   {
      if( strcmp(volumes[i].vpath, path) == 0)
         return i;
   }

   return -1;
}

static dword getHndlFromPath(const char *path)
{
   size_t i;
```

```
   for(i = 0;  i < MAX_OPEN_FILES;  i++)
   {
      if(!strcmp(path, currentProcces->openfiles[i].fpath))
      {
         //if(currentProcces->openfiles[fileHndlr].pos  + offset <= filrSize)
            return currentProcces->openfiles[i].pos;
         //else
            //return 0;
      }
   }
   return 0;
}

static void splitPath(const char *path, char *vol, char *file)
{
   strncpy(vol, path, 2);
   vol[2] = '\0';
   strncpy(file, path + 3, strlen(path + 3));  /**3rd char: / ignored**/
   file[strlen(path + 3)] = '\0';
}
```

Αρχείο: vga.c

```
#include "vga.h"
#include "io.h"

/**??vga must have write() read() */
```

```
void writeToControllReg(byte value, byte mask, byte index);
byte readFromAttributeReg(byte index);
byte getVgaMode();
size_t getMaxColumn();
size_t getMaxRow();
byte fullScreen;

vga_state vga_stt;

void init_vga(void)
{
   vga_stt.vga_mode = getVgaMode();

   clearScreen();
}

byte* getTextBufferAddress()
{
   return (byte*) 0xb8000;
}

void clearScreen()
{
  byte *textBufer;
  size_t rows, columns, i;

  rows = getMaxRow();
  columns = getMaxColumn();
  textBufer = getTextBufferAddress();

  for(i = 0;   i < rows * columns;  i++)
  {
```

```
    *textBufer = ' ';
    *(textBufer + 1) = 0x8F;
    textBufer +=2;
  }

  setCursorPosition((word) 0, (word) 0 );

}



void disableCursor(void)
{
    writeToControllReg(32, 32, CURSOR_START_REGISTER);
}



void enableCursor(void)
{
    writeToControllReg(0, 32, CURSOR_START_REGISTER);
}



void writeToControllReg(byte value, byte mask, byte index)
{
    byte addressRegVal;
    byte dataRegVal;

    INB(addressRegVal, CRTC_ADDRESS_REGISTER);
    OUTB(index, CRTC_ADDRESS_REGISTER);
```

```
   INB( dataRegVal, CRTC_DATA_REGISTER);

   dataRegVal &= (~mask);
   value &= mask;
   dataRegVal |= value;

   OUTB(dataRegVal,CRTC_DATA_REGISTER);
   OUTB(addressRegVal, CRTC_ADDRESS_REGISTER);



}



byte readFromControllReg(byte index)
{
   byte addressRegVal;
   byte dataRegVal;

   INB(addressRegVal, CRTC_ADDRESS_REGISTER);
   OUTB(index, CRTC_ADDRESS_REGISTER);
   INB( dataRegVal, CRTC_DATA_REGISTER);

   OUTB(addressRegVal, CRTC_ADDRESS_REGISTER);

   return dataRegVal;
}



void setCursorPosition(word row, word column)
```

```
{
   byte c_l_low, c_l_high;
   word address = row * getMaxColumn() + column;

   c_l_low = (byte) address & 0xFF;
   c_l_high = (byte) (address >> 8);

   writeToControllReg(c_l_low, 0xFF, CURSOR_LOCATION_LOW);
   writeToControllReg(c_l_high, 0xFF, CURSOR_LOCATION_HIGH);

   fullScreen = 0;
}



void getCursorPosition(cPosition *pos)
{
   word position;

   position =(word) readFromControllReg(CURSOR_LOCATION_LOW);
   position |= ( (word) readFromControllReg(CURSOR_LOCATION_HIGH) ) << 8;

   pos->column = position % getMaxColumn();
   pos->row = position / getMaxColumn();

}

void vga_write(byte *buffer)
{
    __asm volatile("cli;");

   byte *textBuffer = getTextBufferAddress();
```

```
   size_t i, size = getMaxColumn() * getMaxRow() * 2;

   for(i = 0;   i < size;  i++)
      *textBuffer++ = buffer[i];

    __asm volatile("sti;");
}

void printChar(char charToPrint, byte color)
{
return;
   byte *textBuffer = getTextBufferAddress();
   byte *writeAdd = textBuffer;
   cPosition curPos;

   if(fullScreen != 0)
   {
      clearScreen();
      fullScreen = 0;
   }

   getCursorPosition(&curPos);
   if(charToPrint == '\n')
   {
      if(curPos.row >= getMaxRow())
         clearScreen();
      else
         setCursorPosition( curPos.row + 1 , 0);
   }
   else
   {
      //if( (curPos.row >= getMaxRow() - 1) && (curPos.column >= getMaxColumn() - 1) )
```

```
      //    clearScreen();          ///!!! A scroll down function is better!!!

      writeAdd += ((curPos.row * getMaxColumn()) + curPos.column) * 2;

      *writeAdd = charToPrint;
      *(writeAdd + 1) = color;

      if(curPos.column >= getMaxColumn() - 1 && curPos.row < getMaxRow() - 1)
         setCursorPosition(curPos.row + 1, 0);
      else if(curPos.column >= getMaxColumn() - 1 )
         fullScreen = 1;
      else
         setCursorPosition(curPos.row, curPos.column + 1);
   }


}

void printString2(char *string, byte color)   /**??check if buffer overflows*/
{
   byte *textBuffer = getTextBufferAddress();
   cPosition curPos;

   getCursorPosition(&curPos);

   textBuffer +=  ((curPos.row * getMaxColumn()) + curPos.column) * 2;

   while(*string != '\0')
   {

      switch(*string)
      {
```

```
            case '\n':
               textBuffer += getMaxColumn() * 2;
               break;
            case '\t':
               textBuffer += 8;
               break;
            default:
               *textBuffer = *string;
               textBuffer++;
               *textBuffer = color;
               textBuffer++;
               break;
         }


         string++;
      }

}

byte readFromAttributeReg(byte index)
{
   byte regData, regAddress, tmp;

   INB(tmp, INPUT_STATUS_1_REGISTER);
   INB(regAddress, ADDRESS_ATTRIBUTE_REGISTER);
   OUTB(index, ADDRESS_ATTRIBUTE_REGISTER);
   INB(regData, DATA_READ_ATTRIBUTE_REGISTER);
   INB(tmp, INPUT_STATUS_1_REGISTER);
   OUTB(regAddress, ADDRESS_ATTRIBUTE_REGISTER);

   return regData;
```

```
}



byte getVgaMode()
{
   return readFromAttributeReg(MODE_CONTROL_REG_INX);

}

size_t getMaxRow()
{


   switch( vga_stt.vga_mode )
   {
      case TEXT_MODE_80x25:
         return (size_t) 25;
         break;
      default:
         return 0;

   }
}

size_t getMaxColumn()
{
   switch( vga_stt.vga_mode )
   {
      case TEXT_MODE_80x25:
         return (size_t) 80;
         break;
```

```
        default:
            return 0;

    }
}
```

Αρχείο:volume.c

```
#include "volume.h"
#include "stdiok.h"
#include "vfs.h"
#include "string.h"
#include "memory.h"
```

```
#include "fat32.h"



mbrEntry mbr[MAX_MBR_ENTRIES];

volume volumes[MAX_VOLUMES_COUNT];

static dword addVolume( const char*, dword, dword, const char* );
static bool verifyEntry(mbrEntry *);

dword getFsIndx(const char *vol)
{
   size_t i;

   for(i = 0;  i < MAX_VOLUMES_COUNT;  i++)
   {
      if(!strcmp(vol, volumes[i].vpath ) )
      {
            return volumes[i].fs_indx;
      }
   }
   return 0;
}

static bool verifyEntry(mbrEntry *entry)
{
   dword cyl, head, sec;
   unsigned long long totSectors;
printk("Pass test!!\n");
   if(entry->isBootable != PARTITION_IS_BOOTABLE && entry->isBootable !=
PARTITION_NOT_BOOTABLE)
      return false;
printk("Pass test2!!\n");
```

```
   if(entry->lbaStart == 0 || entry->lbaLength == 0)
      return false;
   if(entry->partitionType == 0)
      return false;

   LBA_TO_CHS(entry->lbaStart, cyl, head, sec);

   if(entry->lbaStart >= CHS_MAX_SECTORS - 1)
   {
      if(entry->chsStartCylinder != 1023 || entry->chsStartHead != 254  || entry->chsStartSector
!= 63)
         return false;
   }
   else
   {
      LBA_TO_CHS(entry->lbaStart, cyl, head, sec);

      if(entry->chsStartCylinder != cyl || entry->chsStartHead != head  || entry->chsStartSector !=
sec)
         return false;
   }

   if(entry->lbaLength + entry->lbaStart - 1 >= CHS_MAX_SECTORS - 1)
   {

      if(entry->chsEndCylinder != 1023 || entry->chsEndHead != 254  || entry->chsEndSector !=
63)
         return false;
   }
   else
   {
      totSectors = entry->lbaStart + entry->lbaLength - 1;
      LBA_TO_CHS(totSectors, cyl, head, sec);
```

```
      if(entry->chsEndCylinder != cyl || entry->chsEndHead != head  || entry->chsEndSector != sec)
         return false;
   }

   if(entry->chsStartCylinder > entry->chsEndCylinder || (entry->chsStartCylinder == entry->chsEndCylinder && entry->chsStartHead > entry->chsEndHead)
      || ( entry->chsStartCylinder == entry->chsEndCylinder && entry->chsStartHead == entry->chsEndHead && entry->chsStartSector > entry->chsEndSector)  )
      return false;

   return true;
}

void partitionCreate(const char *disk, dword lbaStart, size_t length)
{

}

dword VolumeRead(const char *path, size_t offset, size_t count, byte *buffer)
{
   size_t i;
   //check if offset + count ecced bounds

   for(i = 0;  i < MAX_VOLUMES_COUNT;  i++)
   {
      if( !strcmp(volumes[i].vpath, path) )
      {
         if( (offset + count ) < (volumes[i].partitionSize * 512 ) )
            return vfread(volumes[i].deviceId, NULL, (offset + volumes[i].lbaStart * 512 ), count, (void*)buffer);
         else
```

```
            return 0;
        }
    }

    return 0;
}

dword VolumeWrite(const char *path, size_t offset, size_t count, byte *buffer)
{
    size_t i;
    //check if offset + count ecced bounds

    for(i = 0;  i < MAX_VOLUMES_COUNT;  i++)
    {
        if( !strcmp(volumes[i].vpath, path) )
        {
            if( (offset + count ) < (volumes[i].partitionSize * 512) )
            {
                //printk("writing at offsest %x\n", (dword) (offset + volumes[i].lbaStart));
                return vfwrite(volumes[i].deviceId, (offset + volumes[i].lbaStart * 512 ), count,
(void*)buffer, 0);
            }
            else
                return 0;
        }
    }

    return 0;
}

int volumeSearch(const char *disk)
{
    mbrEntry mbrentry;
```

```
  dword *mbrLba = NULL;
  size_t i, partition_offset = PARTITION_1_ENTRY_OFFSET;
  static unsigned char label = 'A';
  char volpath[MAX_VPATH_LENGTH];



  byte *mbrbuffer = pageFrameAlloc(1);
  if(mbrbuffer == NULL)
  {
     printk("memory allocation failed\n");
     return -1;
  }

  vfread(disk, NULL, 0, 512, mbrbuffer);
  /*check if vfread fail and return*/
  for(i = 0;  i < 10000; i++);/*wait ide*/

  for(i = 0;  i < 4;  i++)
  {
     mbrentry.isBootable = mbrbuffer[partition_offset + PARTITION_IS_BOOTABLE_OFFSET];


     mbrentry.chsStartHead = mbrbuffer[partition_offset + START_SECTOR_CHS_OFFSET];
     mbrentry.chsStartSector = ( mbrbuffer[partition_offset + START_SECTOR_CHS_OFFSET + 1]
& 0x3F);
     mbrentry.chsStartCylinder = ( ( ((word) mbrbuffer[partition_offset +
START_SECTOR_CHS_OFFSET + 1]) & 0xC0) << 2 ) | ( (dword) (mbrbuffer[partition_offset +
START_SECTOR_CHS_OFFSET + 2]) );


     mbrentry.partitionType = mbrbuffer[partition_offset + PARTITION_TYPE_OFFSET];


     mbrentry.chsEndHead = mbrbuffer[partition_offset + LAST_SECTOR_CHS_OFFSET];
     mbrentry.chsEndSector = ( mbrbuffer[partition_offset + LAST_SECTOR_CHS_OFFSET + 1] &
0x3F);
```

```
      mbrentry.chsEndCylinder = ( ( ((word) mbrbuffer[partition_offset +
LAST_SECTOR_CHS_OFFSET + 1]) & 0xC0) << 2 ) | ( (dword) (mbrbuffer[partition_offset +
LAST_SECTOR_CHS_OFFSET + 2]) );

      mbrLba = (dword*) (mbrbuffer + partition_offset + START_SECTOR_LBA_OFFSET);
      mbrentry.lbaStart = *mbrLba;
      mbrLba += 1;

      mbrentry.lbaLength = *mbrLba;

      if(verifyEntry(&mbrentry) == true)
      {
         volpath[0] = label;
         volpath[1] = ':';
         volpath[2] = '\0';
         addVolume(volpath, mbrentry.lbaStart, mbrentry.lbaLength, disk);

         label++;
      }
      else
      {
         printk("volume is not formated\n");
      }

      partition_offset += PARTITION_ENTRY_SIZE;
   }

   pageFrameFree((dword)mbrbuffer / PAGE_FRAME_SIZE, 1);

   return 0;
}
```

```
static dword addVolume( const char *volpath, dword lbastart, dword lbacount, const char
*deviceId )
{
   size_t i;

   for(i = 0;  i < MAX_VOLUMES_COUNT;  i++)
   {
      if(volumes[i].vpath[0] == '\0')
      {
         volumes[i].lbaStart = lbastart;
         volumes[i].partitionSize = lbacount;
         strcpy(volumes[i].vpath, volpath);
         strcpy(volumes[i].deviceId, deviceId);

         printk("vpath: ");
         printk(volumes[i].vpath);
         printk(" at lba %x\n",  lbastart);

         volumes[i].fs_indx = installFat32(volpath);

         return 0;
      }
   }

   return 1;
}

void initVolumes(void)
{
   size_t i;

   for(i = 0;  i < MAX_VOLUMES_COUNT;  i++)
   {
```

```
        volumes[i].vpath[0] = '\0';
        volumes[i].fs_indx = 0;
    }
}
```

Αρχείο: waitProc.c

```
#include "waitProc.h"
#include "procCondition.h"
#include "proccessStruct.h"
#include "process.h"
#include "stdiok.h"
```

```
#include "ProcCondition.h"

extern pcbNode *currentProcces, *readyProccesQueue, *readyProccesQueueTail,
*blockedProccesQueue;

#define HEAD       0x1
static void changeProccessQueue(dword pos, pcbNode **sourceList, pcbNode **destList)
{
   pcbNode *temp;

   switch(pos){
   case HEAD:
      temp = *sourceList;
      *sourceList = (*sourceList)->next;
      temp->next = *destList;
      *destList = temp;
      break;
   }
}

static void blockProcces(void)
{
   currentProcces->state = blocked;
   changeProccessQueue(HEAD, &readyProccesQueue, &blockedProccesQueue);

   if(readyProccesQueue == NULL)
      readyProccesQueueTail = NULL;


   pcbNode *curr = blockedProccesQueue;

   while(curr != NULL)
      curr = curr->next;
```

```
   //currentProcces = NULL;
   __asm volatile("int $32;");
}

/*check if an alredy finished proccess exist in list*/
static pcbNode* terminatedChildExist(pcbNode *proc, int *status)
{
   childProccesNode *head = proc->childPidListHead;
   childProccesNode *curr = head;

   while(curr != NULL)
   {
      if(curr->hasReturn == true)
      {
         *status = curr->retVal;
         return curr->pcb;
      }
      curr = curr->next;
   }
  return NULL;
}

int condWait(cond_t *cond)
{
   if(currentProcces == NULL)
      return -1;

   cond_add_proc(cond, currentProcces);
   blockProcces();
}
```

```
int wait(int *status)
{
   pcbNode *proc = NULL;

   if(currentProcces != NULL)
   {
      if(currentProcces->childPidListHead == NULL)
         return -1;

      if((proc = terminatedChildExist(currentProcces, status)) != NULL)
      {
         return proc->pid;
      }

      blockProcces();

      if((proc = terminatedChildExist(currentProcces, status)) != NULL)
      {
         return proc->pid;
      }
   }
   return -1;
}
```

Αρχείο: createProcces.h

```
#ifndef _CREATE_PROCCES_H
#define _CREATE_PORCCES_H

#include "type.h"
#include "proccessStruct.h"
```

```
typedef struct Sproc_mem_regions{
   byte *text, *bss, *data, *stack, *kstack;
   size_t textsize, bsssize, stacksize, kstacksize; /*size in pages*/
   void* entry;
} proc_mem_regions;

dword createProcces(const char *exepath);
void addDriver(const char *exepath);
void sendMessage(proccesMsg *msg);
dword ScgetMessage(proccesMsg *msg);
dword ScsetProcProperties(dword procDescr);
dword ScsetActiveProc(dword procHndl);
dword ScLoadDriver(const char *driverpath);

#endif // _CREATE_PROCCES_H
```

Αρχείο: criticalError.h

```
#ifndef _CRITICAL_ERROR_H
#define _CRITICAL_ERROR_H

#include "type.h"

void criticalError(const char msg[], dword);
```

```
#endif // _CRITICAL_ERROR_H
```

Αρχείο: destroyProccess.h

```
#ifndef _DESTROY_PROCCESS_H
#define _DESTROY_PROCCESS_H

#include "type.h"
#include "proccessStruct.h"
```

```
void unblockProccess(pcbNode *pcb);

#endif // _DESRTOY_PROCCESS_H
```

Αρχείο: device.h

```
#ifndef DEVICE_H
#define DEVICE_H

#include "type.h"

#define MAX_NAME_STRING      128
```

```
#define MAX_ID_STRING       128
#define MAX_PATH_ID_STRING   32

/*DEVICE TYPES*/
#define NO_DEVICE      0x00
#define STORAGE        0x01



#include "vFileOperations.h" /**struct operations*/



typedef struct Sdevice{
   byte type;
   char deviceName[MAX_NAME_STRING];
   char deviceId[MAX_ID_STRING];
   char pathId[MAX_PATH_ID_STRING];
   operations op;

} device;

#define MAX_DEV_SUPPORTED   32

extern device devices[];

/*Error codes for addDevice*/
#define NO_FREE_SLOT_FOR_DEVICE    -1
int addDevice(const char* deviceId, const char* deviceName, byte devType, operations *op);

#endif // DEVICE_H
```

Αρχείο: exeptions.h

```
#ifndef _EXEPTIONS_H
#define _EXEPTIONS_H

#include "interupts.h"

/*Intel mnemonics vector nubmer for exceptions
```

```
#define DE     0
#define DB     1
#define BP     3
#define OF     4
#define BR     5
#define UD     6
#define NM     7
#define DF     8
#define TS     10
#define NP     11
#define SS     12
#define GP     13
#define PF     14
#define MF     16
#define AC     17
#define MC     18
#define XF     19
*/

#define INTR_HNDL __attribute__((interrupt, no_caller_saved_registers))

INTR_HNDL void divZeroEx(struct interrupt_frame*);
INTR_HNDL void debugEx(struct interrupt_frame*);
INTR_HNDL void breakpointEx(struct interrupt_frame*);
INTR_HNDL void overflowEx(struct interrupt_frame*);
INTR_HNDL void boundEx(struct interrupt_frame*);
INTR_HNDL void invalodOpCodeEx(struct interrupt_frame*);
INTR_HNDL void deviceNotAvailEx(struct interrupt_frame*);
INTR_HNDL void doubleFaultEx(struct interrupt_frame*, dword);
INTR_HNDL void coprocessorOverrunEx(struct interrupt_frame*);
INTR_HNDL void tssEx(struct interrupt_frame*, dword);
INTR_HNDL void segmentNotPresentEx(struct interrupt_frame*, dword);
```

```
INTR_HNDL void StackFaultEx(struct interrupt_frame*, dword);
INTR_HNDL void generalProtectionFaultEx(struct interrupt_frame*, dword);
INTR_HNDL void PageFaultEx(struct interrupt_frame*, dword);
INTR_HNDL void x87FPUErrorEx(struct interrupt_frame*);
INTR_HNDL void AligmentCheckEx(struct interrupt_frame*, dword);
INTR_HNDL void MachineCheckEx(struct interrupt_frame*);
INTR_HNDL void SIMDFloatPointEx(struct interrupt_frame*);


#endif // _EXEPTIONS_H
```

Αρχείο: exportedSymbols.h

```
#ifndef _EXPORDER_SYMBOLS_H
#define _EXPORDER_SYMBOLS_H

#include "type.h"
```

```
#define MAX_EPRORTED_SYMBOLS    0x10

typedef struct Ssymbol{
   dword address;
   char name[30];
} symbol;

extern symbol symbols[MAX_EPRORTED_SYMBOLS];
void initExportedSymbols(void);

#define EXPORT_SYMBOL(sym, pos)  symbols[pos].address = (dword) sym //symbols[pos].name
= (dword) sym;

#endif  /**_EXPORDER_SYMBOLS_H*/
```

Αρχείο: fat32.h

```
#ifndef _FAT32_H
#define _FAT32_H

#include "type.h"
#include "fat32dir.h"

/*volumeId offsets**/
```

```
#define OFFSET_BYTES_PER_SECTOR    0x0B
#define OFFSET_SECTORS_PER_CLUSTER  0x0D
#define OFFSET_RESERVED_SECTORS    0x0E
#define OFFSET_FATS_COUNT          0x10
#define OFFSET_SECTORS_PER_FAT     0x24
#define OFFSET_ROOT_DIR_CLUSTER    0x2C
#define OFFSET_SIGNATURE           0x1FE

#define FATS_COUNT                 0x02
#define BYTES_PER_SECTOR           0x512
#define SIGNATURE                  0x55AA  /*Little endian*/
//#define VALIDATE_SECTORS_PER_CLUS(spc)  ( ((spc) < 128) * () )
//#define VALIDATE_SECTORS_PER_CLUS(spc)  ((spc) - 128) *

#define FAT_INDX_TO_LBA(indx)          ((indx)  >>  0x8 )
#define FAT_INDX_TO_OFFSET(indx)       ((indx)  &  0xFF )

#define END_OF_FAT_LIST        0x0FFFFFFF
#define FREE_FAT_ENTRY         0x00000000

#define     SEARCH_DIR     0x1
#define     SEARCH_FILE    0x2
#define     SEARCH_ALL     (SEARCH_DIR | SEARCH_FILE)

#define IS_LO_CA_CHAR(ch)       ((ch) >= 97  && (ch) <= 122)
#define IS_UP_CA_CHAR(ch)       ((ch) >= 65  && (ch) <= 90)
#define IS_ASCII_CHAR(ch)       ( IS_LO_CA_CHAR(ch) || IS_UP_CA_CHAR(ch) )

#define IS_ASCII_NUM(ch)        ((ch) >= 48 && (ch) <= 57 )
#define IS_SPEC_FAT_CHAR(ch)    ((ch) == '$' || (ch) == '%' || (ch) == '\'' || (ch) == '-' || (ch) ==
'@' || (ch) == '~' || (ch) == '`' || (ch) == '!' || (ch) == '(' ||  \
                        (ch) == ')' || (ch) == ')' || (ch) == '{' || (ch) == '}' || (ch) == '#' || (ch) == '&' ||
(ch) == '.' )
```

```
#define IS_VALID_SFN_CHAR(ch)   ( IS_ASCII_CHAR(ch) || IS_ASCII_NUM(ch) ||
IS_SPEC_FAT_CHAR(ch) )
#define TO_UP_CA_ASCII(ch)      ( (ch) - 32)



typedef struct SfatCache
{
  size_t sizeSectors, cacheHit;  /**countSectors better name*/
  dword *buffer;
  dword startlba;
  bool isActive;
  byte dirtyBit;
} fatCache;

typedef struct Scluster{
   size_t index;
   byte *data;
   byte dirtyBit;
} cluster;

#define GLOBAL_CLUSTER_MAX_RECORDS  0x10

typedef struct SclusterCache{
   size_t recordsCount, clusterSize;
   cluster clusters[GLOBAL_CLUSTER_MAX_RECORDS];
} clusterCache;

#define GLOBAL_CACHE_SEC_SIZE   0x10

typedef struct Sf32_state{
   bool isformated;
   byte sectorsPerCluster, fatsCount;
```

```
   word bytesPerSector, reservedSectors;
   dword sectorsPerFat, rootCluster;
   //fatBuffer genFatBuff;
   char vPath[255];
   fatCache globalFatCache;
   clusterCache globalClusterCache;
} f32_state;

typedef struct Sfile32{
   fatCache fCache;
   clusterCache clCache;
   size_t firstCluster;
   size_t currCluster;
   size_t currClusterOffset;
   size_t fSize;
   size_t pos;
   size_t fdescrClIndx;
   size_t fdescrClOffset;
   bool eof;
} file32;

static inline dword fat_offset_bits(size_t bytesPerSec)
{
   /**Alternative return (logBase2(bytesPerSec / sizeof(dword)) )  bytesPerSec >= 512  &&
bytesPerSec <= 16384 and bytesPerSec is power of 2*/

   switch(bytesPerSec)
   {
   case (dword) 512:
      return 7;
   case (dword) 1024:
      return 8;
   case (dword) 2048:
```

```
      return 9;
   case (dword) 4096:
      return 10;
   case (dword) 8192:
      return 11;
   case (dword) 16384:
      return 12;
   }

   return (dword) 0xFFFFFFFF;
}
static inline dword fat_indx_to_lba(dword indx, size_t bytesPerSec)
{
   return (0xFFFFFFF & indx) >> fat_offset_bits(bytesPerSec);
}
bool init(const char *);
dword locateFile(const word *, dword, dword * , dword *, size_t *, bool *);

#define FS_MAX_OPEN_FILES       1024
#define INVALID_FILE_HANDLER  (dword) 0xFFFFFFFF

#endif // _FAT32_H
```

Αρχείο: fat32dir.h

```
#ifndef _FAT32_DIR_H
#define _FAT32_DIR_H

#include "type.h"

#define ATTR_READ_ONLY     0x01
#define ATTR_HIDDEN        0x02
#define ATTR_SYSTEM        0x04
```

```
#define ATTR_VOLUME_ID     0x08

#define ATTR_DIRECTORY     0x10

#define ATTR_ARCHIVE       0x20

#define ATTR_LONG_NAME     ATTR_READ_ONLY | ATTR_HIDDEN | ATTR_SYSTEM |
ATTR_VOLUME_ID


#define ENTRY_IS_READ_ONLY(directoryEntry)      ( (directoryEntry.data)[11] &
ATTR_READ_ONLY )

#define ENTRY_IS_HIDDEN(directoryEntry)         ( (directoryEntry.data)[11] & ATTR_HIDDEN )

#define ENTRY_IS_SYSTEM(directoryEntry)         ( (directoryEntry.data)[11] & ATTR_SYSTEM )

#define ENTRY_IS_VOLUME_ID(directoryEntry)      ( (directoryEntry.data)[11] &
ATTR_VOLUME_ID )

#define ENTRY_IS_DIRECTORY(directoryEntry)      ( (directoryEntry.data)[11] & ATTR_DIRECTORY
)

#define ENTRY_IS_ARCHIVE(directoryEntry)        ( (directoryEntry.data)[11] & ATTR_ARCHIVE )

#define ENTRY_IS_LONG_NAME(directoryEntry)      ( ((directoryEntry.data)[11] &
(ATTR_LONG_NAME)) == (ATTR_LONG_NAME) )

#define ENTRY_IS_FILE(directoryEntry)          (    !ENTRY_IS_DIRECTORY(directoryEntry)    &&
!ENTRY_IS_VOLUME_ID(directoryEntry)    )


#define DIR_GET_ATTRIBUTE_BYTE(directoryEntry)    ( (directoryEntry.data)[11] )

#define DIR_GET_CLUSTER_NUM(directoryEntry)      ( (*(word*)(&((directoryEntry.data)[20])) <<
16 ) |  (*(word*)(&((directoryEntry.data)[26])))  )

#define DIR_GET_FILE_SIZE(directoryEntry)        ( *(dword*)(&((directoryEntry.data)[28])) )

#define DIR_GET_WRTDATE(directoryEntry)          ( *(word*)(&((directoryEntry.data)[24])) )

#define DIR_GET_WERTTIME(directoryEntry)         ( *(word*)(&((directoryEntry.data)[22])) )


#define DIR_ENTRY_IS_FREE(directoryEntry)        ( (directoryEntry.data)[0] == 0xE5 )

#define DIR_EOF(directoryEntry)                  ( (directoryEntry.data)[0] == 0x00 )


#define DIR_NAME_MAIN_PART_MAX_CHARS     0x8

#define DIR_NAME_EXTENSION_MAX_CHARS     0x3

#define DIR_NAME_MAX_CHARS               (DIR_NAME_MAIN_PART_MAX_CHARS +
DIR_NAME_EXTENSION_MAX_CHARS)
```

```
#define DIR_INVALID_NAME_CHARS 0x22, 0x2A, 0x2B, 0x2C, 0x2E, 0x2F, 0x3A, 0x3B, 0x3C,
0x3D, 0x3E, 0x3F, 0x5B, 0x5C, 0x5D, 0x7C

#define DIR_ENTRY_SPACE_CHAR     0x20
#define DIR_KANJI_LEAD_BYTE      0x05

#define LDIR_NAME1  0x01
#define LDIR_NAME2  0x0E
#define LDIR_NAME3  0x1C

#define LDIR_NAME1_LAST  0x09
#define LDIR_NAME2_LAST  0x18
#define LDIR_NAME3_LAST  0x1E

#define LDIR_GET_ORDER(directoryEntry)  ( (directoryEntry.data)[0] & 0x3F  )

#define DIR_ENTRY_SIZE      32

typedef struct SdirecoryEntry{
   unsigned char data[DIR_ENTRY_SIZE];
} directoryEntry;

#define DIRECTORY_CLUSTER_BUFFER_SIZE   0x10
#define MAX_PATH_COMPNENT_CHARS  260

typedef struct Sdirectory{
   dword startCluster;
   dword currentCluster;
   dword nextCluster;
   size_t data_buff_size;
   size_t CountClusters;
   size_t currEntryIndex;
   word curr_lnf[MAX_PATH_COMPNENT_CHARS];
```

```
    directoryEntry *curr_entry;
    byte *data;
    bool eod;
    size_t ClOffset;
    dword clIndx;
    dword loadedFats[DIRECTORY_CLUSTER_BUFFER_SIZE];
}directory;

//bool dirNameIsValid(directoryEntry);
size_t dirGetLfn(directoryEntry, word*);
word* dirGetName(directoryEntry, word*);

#endif // _FAT32_DIR_H
```

Αρχείο: file.h

```
#ifndef _FILE_H
#define _FILE_H

#include "type.h"
#include "proccessStruct.h"

dword initProcFile(pcbNode *);
dword addFiletoPcb(const char* fpath, size_t fsize, dword fsIndx);
dword updateFileCursor(const char *path, size_t pos);
```

```
const char *getPathFromHandle(dword fileHndlr);//const char or char const???
size_t getPosFromHandle(dword fileHndlr);//const char or char const???


#endif // _FILE_H
```

Αρχείο: fileSystem.h

```
#ifndef FILE_SYSTEM_H
#define FILE_SYSTEM_H

#include "type.h"

#include "vFileOperations.h" /**struct operations*/


#define MAX_FILE_SYSTEM_NAME 128
```

```
#define MAX_FILE_SYSTEM_ID_STRING 128

/*FS TYPES*/
#define NO_FS   0x00
#define FAT32    0x01

typedef struct SfileSystem{
   byte type;
   char fsName[MAX_FILE_SYSTEM_NAME];
   char fsId[MAX_FILE_SYSTEM_ID_STRING];
   operations op;

} fileSystem;

#define MAX_FS_SUPPORTED 16

extern fileSystem fileSystems[];

int addFileSystem(const char*, const char*, byte, operations *);

#endif // FILE_SYSTEM_H
```

Αρχείο: gui.h

```
#ifndef _GUI_H
#define _GUI_H

#include "type.h"
#include "proccessStruct.h"

#define MAX_GUI_PROCCESES   32
#define NO_SELECTED_PROCCESS 0xFFFFFFFF
```

```
void initVideo(void);

typedef struct SguiProccesesBuf{
   pcbNode *nodes[MAX_GUI_PROCCESES];
   size_t curr, last;
} guiProccesesBuf;

bool procGuiInit(pcbNode *currentProcces);
bool guiNext(pcbNode *currentProcces);
void procGuiDestroy(pcbNode *currentProcces);

extern guiProccesesBuf guiProcceses;

#endif // _GUI_H
```

Αρχείο: ide.h

```
#ifndef _IDE_H
#define _IDE_H

#include "type.h"
#include "io.h"

/*Comand block registers*/
#define DATA_REG            0x1F0
#define ERROR_REG            0x1F1 /*Read only*/
```

```
#define FEATURE_REG          0x1F1  /*Write only*/
#define SECTOR_COUNT_REG      0x1F2
#define SECTOR_NUM_REG        0x1F3
#define CYLINDER_LOW_REG      0x1F4
#define CYLINDER_HIGH_REG     0x1F5
#define DRIVE_HEAD_REG        0x1F6
#define STATUS_REG            0x1F7 /*Read only*/
#define COMMAND_REG           0x1F7 /*Write only*/

/*Control block registers*/
#define DATA_BUS_HIGH_IMP_REG   0x3F0 /*Read only. 3F0 - 3F5*/
#define ALTERNATE_STATUS_REG    0x3F6 /*Read only*/
#define DEVICE_CONTROL_REG      0x3F6 /*Write only*/
#define DRIVE_ADDRES_REG        0x3F7 /*Read only*/

/*ATA commands*/
#define IDENTIFY_DRIVE        0xEC
#define READ_SECTORS          0x21
#define READ_SECTORS_RETRY    0x20
#define WRITE_SECTORS         0x31
#define WRITE_SECTORS_RETRY   0x30

typedef struct chs{
   word cylinder;
   word head:4;
   word sector:8;    /*sectors per track*/
} chs_t;

typedef struct lba{
   byte low, mid, high;
} lba;
```

```
/*Convert CHS to LBA
C: cylinder SpT: sectors per track
H: head SpT: sectors per track S:sectors*/
#define toLba(C, SpC, H, SpT, S)  ((C) * (SpC) + (H) * (SpT) + (S) - 1)

static inline word readDataReg(void){
   word val;

   INW(val, DATA_REG);
   return val;
}

#define MAKE_READ_REGISTER(name, rettype, reg)      \
static inline rettype read##name##Reg(void){        \
   rettype val;                              \
   INB(val, reg);                             \
   return (rettype) val;                          \
}

MAKE_READ_REGISTER(DriveHead, byte, DRIVE_HEAD_REG)
MAKE_READ_REGISTER(Error, byte, ERROR_REG)
MAKE_READ_REGISTER(SectorCount, byte, SECTOR_COUNT_REG)
MAKE_READ_REGISTER(AlternateStatus, byte, ALTERNATE_STATUS_REG)
MAKE_READ_REGISTER(Status, byte, STATUS_REG)
MAKE_READ_REGISTER(SectorNum, byte, SECTOR_NUM_REG)
MAKE_READ_REGISTER(CylinderLow, byte, CYLINDER_LOW_REG)
MAKE_READ_REGISTER(CylinderHigh, byte, CYLINDER_HIGH_REG)
MAKE_READ_REGISTER(DriveAddres, byte, DRIVE_ADDRES_REG)
MAKE_READ_REGISTER(DataBusHighImp, byte, DATA_BUS_HIGH_IMP_REG)

#undef MAKE_READ_REGISTER
```

```
static inline void writeDataReg(word val){
   OUTW(val, DATA_REG);
}


#define MAKE_WRITE_REGISTER(name, reg, type0)            \
static inline void write##name##Reg(type0 val){          \
   OUTB(val, reg);                                       \
}

MAKE_WRITE_REGISTER(DriveHead, DRIVE_HEAD_REG, byte)
MAKE_WRITE_REGISTER(SectorCount, SECTOR_COUNT_REG, byte)
MAKE_WRITE_REGISTER(SectorNum, SECTOR_NUM_REG, byte)
MAKE_WRITE_REGISTER(Feature, FEATURE_REG, byte)
MAKE_WRITE_REGISTER(CylinderLow, CYLINDER_LOW_REG, byte)
MAKE_WRITE_REGISTER(CylinderHigh, CYLINDER_HIGH_REG, byte)
MAKE_WRITE_REGISTER(DeviceControl, DEVICE_CONTROL_REG, byte)
MAKE_WRITE_REGISTER(Command, COMMAND_REG, byte)

#undef MAKE_WRITE_REGISTER

#define MASTER     0
#define SLAVE      (1 << 4)
static inline void selectDrive(byte drive)
{
   writeDriveHeadReg( (byte)(0xA0 | drive) );
}

#define CHS     0
#define LBA      (1 << 6)
static inline void setAddressMode(byte drive)
```

```
{
   writeDriveHeadReg( (byte)(0xA0 | drive) );
}

static inline void resetDrive(void)
{
   OUTB(0xE, DEVICE_CONTROL_REG);
}

static inline void setCHS(chs_t *chs)
{
   writeCylinderHighReg( (byte) (chs->cylinder >> 8) );
   writeCylinderLowReg( (byte) (chs->cylinder & 0xFF) );
   writeDriveHeadReg( (byte) (chs->head) | (readDriveHeadReg() & 0xF0) );
   writeSectorNumReg( (byte) (chs->sector) );
}

static inline void getCHS(chs_t *chs)
{
   chs->cylinder = ((word) readCylinderHighReg() << 8);
   chs->cylinder |= (word) readCylinderLowReg();
   chs->head = (word) (readDriveHeadReg() & 0x0F);
   chs->sector = (word) readSectorNumReg();
}

static inline void setLba(dword lba)
{
   writeCylinderHighReg( (byte) ((lba >> 16) & 0xFF) );
   writeCylinderLowReg( (byte) ((lba >> 8) & 0xFF) );
   writeDriveHeadReg( (byte) (((lba >> 24) & 0xF) | (readDriveHeadReg() & 0xF0) ) );
   writeSectorNumReg( (byte)  (lba & 0xFF) );
}
```

```
static inline dword getLba(void)
{
   dword lba = 0;

   lba = ((dword) readCylinderHighReg() << 16);
   lba |= ((dword) readCylinderLowReg() << 8);
   lba |= ((dword) readDriveHeadReg() & 0xF) << 24;
   lba |= (dword) readSectorNumReg();

   return lba;
}

typedef struct diskInf{
   word cyl_cnt;
   word head_cnt;
   word spt_cnt;   /*sectors per track*/
   char serial_num[20];
   word buffer_sz; /*zero means parametre not specified*/
   char model_num[40];
   bool dword_io_supp;
   bool is_lba_supp;
   bool is_dma_supp;
   dword total_sectors; /*lba only*/

} diskInfo;

void initDisk(void);
void installDevice(void);
size_t readSectors(dword add, size_t count, void *buffer); /*read n sectors with chs*/
int writeSectors(dword lba, size_t count, void *buffer);
dword getSize(void);
```

```
#endif // _IDE_H
```

Αρχείο: interupts.h

```
#ifndef _INTERUPTS_H
#define _INTERUPTS_H

#include "type.h"


#define DPL0       0x00
#define DPL3       (0x03 << 5)
```

```
#define TRAP_GATE   0x8F
#define INT_GATE    0x8E

#pragma pack(1)

 struct interrupt_frame{
    dword ip;
    dword cs;
    dword flags;
    dword sp;
    dword ss;
};

struct interuptGate{
    u16bit offsetLow16;
    word   segmentSelector;
    byte   zero;
    byte   typeAttr;
    u16bit offsetHigh16;

} __attribute__((packed));

struct idtr{
    word limit;
    dword base;
} __attribute__((packed)) interuptDescTableReg;

#pragma pack(0)

void addIntGate(size_t index, word cs,dword offset, byte flags);
void initIdt(void);
```

```
#endif // _INTERUPTS_H
```

Αρχείο: io.h

```
#ifndef _IO_H
#define _IO_H

#include "type.h"


#define OUTB(a, port) __asm volatile("outb %%al, %%dx;"                    \
                                     :                                     \
                                     : "d" ((dword) port ), "a" ((byte) a ) )
```

```
#define OUTW(a, port) __asm volatile("outw %%ax, %%dx;"                    \
                        :                                  \
                        : "d" ((dword) port ), "a" ((word) a ) )

#define INB(a, port) __asm volatile("inb %%dx, %%al;"                      \
                        : "=a" ((byte) a )                 \
                        : "d" ((dword) port ) )

#define INW(a, port) __asm volatile("inw %%dx, %%ax;"                      \
                        : "=a" ((word) a )                 \
                        : "d" ((dword) port ) )

#endif // _IO_H
```

Αρχείο: ioblock.h

```
#ifndef _IO_BLOCK_H
#define _IO_BLOCK_H

#include "type.h"
#include "proccessStruct.h"

/*blocks the current procces for io*/
void ioblock(void);
/*unblock  proc*/
```

```
void iounblock(pcbNode *proc);

#endif // _IO_BLOCK_H
```

Αρχείο: kernelHeap.h

```
#ifndef _KERNELHEAP_H
#define _KERNELHEAP_H

#include "type.h"

#define KERNEL_HEAP_START_ADD   0x120A00
#define KERNEL_HEAP_END_ADD     0x400000

typedef struct SheapNode{
```

```
   struct SheapNode *next;
   size_t size;
   bool isFreeSlot;
} heapNode;

void initKernelHeap(void);
void* kmalloc(size_t size);
void kfree(void* ptr);

#endif // _KERNELHEAP_H
```

Αρχείο: keyboard.h

```
#ifndef _KEYBOARD_H
#define _KEYBOARD_H

#include "type.h"

#define KEYBOARD_DATA_BUFFER    0x60

#define KEY_PRESSED(k)     (!!!((((byte)(k)) & 128)) )     ///Return 1 if key pressed 0 if released
```

```
void keyboardHandler(void);

#endif // _KEYBOARD_H
```

Αρχείο: memory.h

```
#ifndef _MEMORY_H
#define _MEMORY_H

#include "type.h"
#include "proccessStruct.h"

typedef struct SmemoryMapEntry
{
   qword baseAddress;
```

```
   qword regionLength;
   dword regionType;
   dword extendedAttr;
} memoryMapEntry;


#define EXPORT_SYMBOLS_ARRAY_POINTER    (*(dword*)0x100000)    ///Pointer to exported symbol data stucure

#define MEMORY_MAP_ENTRIES_COUNT      (word*)0x100200///Number of entries in memory map table located in this address
#define BIOS_MEMORY_MAP_ADDRESS (memoryMapEntry*) 0x100204///location of memory map table
#define PAGE_FRAME_BITMAP_BASE_ADD      (dword*)0x100A00///used after bios map copy to higher address

#define REG_TYPE_RESERVED   (dword) 2   ///Bios defined
#define REG_TYPE_FREE   (dword) 1

#define PAGE_FRAME_SIZE     4096
#define GET_PAGE_FRAME_FROM_ADDRESS(a)  ( ((dword)(a))/(dword)PAGE_FRAME_SIZE )

#define PAGE_IS_FREE     0x0
#define PAGE_IS_RESERVED 0x1

#define GET_PAGE_STATUS(page) (*(byte*)(((dword)PAGE_FRAME_BITMAP_BASE_ADD) + ((page) / 8)))   &    (128 >> ((page) % 8) )

void initMemoryMap(void);
void printMemoryMap(void);
dword getMemorySize(void);
size_t countFreePhysicalPages(void);
void ReservePageFrames(dword basePgFrame, size_t pgFrameCount);///reserve page frames
void ReleasePageFrames(dword basePgFrame, size_t pgFrameCount);///free page frames
```

```
void* pageFrameAllocBase(size_t frameCnt, dword basePage);
void* pageFrameAlloc(size_t frameCnt);
void pageFrameFree(dword basePage, size_t count);

pageFrameRegionEntry* createProccessCatalog(void);
void destroyProccesCatalog(void);
int ScFreePhysicalPages(void* ptr);
void* ScMemAlloc(size_t length, dword base_address, dword mode, dword protection);

#define     ALLOC_MODE_FIXED_ADD    0x01
#define     ALLOC_MODE_ANY_ADD      0x02

#endif // _MEMORY_H
```

Αρχείο: pe.h

```
#ifndef _PE_H
#define _PE_H

#include "type.h"

#define PE_COFF_HEADER_SZ      0x18

#define PE_FORMAT_PE32        0x10B
#define PE_FORMAT_PE32_PLUS    0x20B
```

```
typedef struct SpeHeader{
   dword peSign, peHdrSign, magicNum;
   size_t peHdrOff, OpHdrSz, OpHdrOff, SecCnt;
   size_t SecTblOff, codeSecOff;
   dword codeSz;
   dword initDSz;
   dword uninDSz;
   dword rvaEntP;
   dword rvaCodeBase;
   dword rvaDataBase;
   dword imagebase;
}peHeader;

void loadProcFromDisk(const char *, dword *, int);

#endif // _PE_H
```

Αρχείο: pic.h

```
/*source: osDev*/
#ifndef _PIC_H
#define _PIC_H

#include "type.h"

#define PIC1_COMMAND 0x20
#define PIC2_COMMAND 0xA0
#define PIC1_DATA    0x21
```

```
#define PIC2_DATA    0xA1

/*** From osDev/8259_PIC ***/
#define ICW1_ICW4          0x01          /* ICW4 (not) needed */
#define ICW1_SINGLE        0x02          /* Single (cascade) mode */
#define ICW1_INTERVAL4          0x04             /* Call address interval 4 (8) */
#define ICW1_LEVEL         0x08          /* Level triggered (edge) mode */
#define ICW1_INIT          0x10          /* Initialization - required! */

#define ICW4_8086          0x01          /* 8086/88 (MCS-80/85) mode */
#define ICW4_AUTO          0x02          /* Auto (normal) EOI */
#define ICW4_BUF_SLAVE          0x08             /* Buffered mode/slave */
#define ICW4_BUF_MASTER    0x0C             /* Buffered mode/master */
#define ICW4_SFNM          0x10          /* Special fully nested (not) */


/*Pic commands*/
#define PIC_EOI   0x20  /*PIC end of interrupt*/


#define DEFAULT_MASTER_PIC_VECTOR_OFFSET    0x20
#define DEFAULT_SLAVE_PIC_VECTOR_OFFSET     0x28

#if (DEFAULT_MASTER_PIC_VECTOR_OFFSET & 0x7) || (DEFAULT_SLAVE_PIC_VECTOR_OFFSET
& 0x7)
#error Pic vector offset is not aligned
#endif // Aligment check

/*Standard ISA irqs*/

#define ISA_PIT_IRQ          0x0
#define ISA_KEYBOARD_IRQ       0x1/*Irq 2 used iternaly by the pic*/
#define ISA_COM2_IRQ          0x3
```

```
#define ISA_COM1_IRQ          0x4
#define ISA_LPT2_IRQ          0x5
#define ISA_FLOPY_DISK_IRQ    0x6
#define ISA_LPT1_IRQ          0x7/*Same irq as spurious interrupt*/
#define ISA_SPURIOS_INT_IRQ   0x7
#define ISA_CMOS_RT_CLCK_IRQ  0x8
#define ISA_SCSI_NIC_IRQ      0x9
#define ISA_SCSI_NIC2_IRQ     0xA
#define ISA_SCSI_NIC3_IRQ     0xB
#define ISA_PS2_MOUSE_IRQ     0xC
#define ISA_FPU_IRQ           0xD
#define ISA_PRIMARY_ATA_IRQ   0xE
#define ISA_SECONDARY_ATA_IRQ 0xF

/*Standard ISA Irqs with offset vector*/
#define PIT_IRQ           ISA_PIT_IRQ  + DEFAULT_MASTER_PIC_VECTOR_OFFSET
#define KEYBOARD_IRQ      ISA_KEYBOARD_IRQ + DEFAULT_MASTER_PIC_VECTOR_OFFSET
#define COM2_IRQ          ISA_COM2_IRQ + DEFAULT_MASTER_PIC_VECTOR_OFFSET
#define COM1_IRQ          ISA_COM1_IRQ + DEFAULT_MASTER_PIC_VECTOR_OFFSET
#define LPT2_IRQ          ISA_LPT2_IRQ + DEFAULT_MASTER_PIC_VECTOR_OFFSET
#define FLOPY_DISK_IRQ    ISA_FLOPY_DISK_IRQ + DEFAULT_MASTER_PIC_VECTOR_OFFSET
#define LPT1_IRQ          ISA_LPT1_IRQ + DEFAULT_MASTER_PIC_VECTOR_OFFSET
#define SPURIOS_INT_IRQ   ISA_SPURIOS_INT_IRQ + DEFAULT_MASTER_PIC_VECTOR_OFFSET
#define CMOS_RT_CLCK_IRQ  ISA_CMOS_RT_CLCK_IRQ +
DEFAULT_MASTER_PIC_VECTOR_OFFSET
#define SCSI_NIC_IRQ      ISA_SCSI_NIC_IRQ + DEFAULT_MASTER_PIC_VECTOR_OFFSET
#define SCSI_NIC2_IRQ     ISA_SCSI_NIC2_IRQ + DEFAULT_MASTER_PIC_VECTOR_OFFSET
#define SCSI_NIC3_IRQ     ISA_SCSI_NIC3_IRQ + DEFAULT_MASTER_PIC_VECTOR_OFFSET
#define PS2_MOUSE_IRQ     ISA_PS2_MOUSE_IRQ + DEFAULT_MASTER_PIC_VECTOR_OFFSET
#define FPU_IRQ           ISA_FPU_IRQ + DEFAULT_MASTER_PIC_VECTOR_OFFSET
#define PRIMARY_ATA_IRQ   ISA_PRIMARY_ATA_IRQ +
DEFAULT_MASTER_PIC_VECTOR_OFFSET
```

```
#define SECONDARY_ATA_IRQ   ISA_SECONDARY_ATA_IRQ +
DEFAULT_MASTER_PIC_VECTOR_OFFSET

#define KEEP_REGS       __attribute__((no_caller_saved_registers))

s32bit remapIrq(byte mstVecOff, byte slvVecOff);
KEEP_REGS void PicEoi(byte irq);
void PicSetIrqMask(byte irqLine);
void PicClearIrqMask(byte irqLine);
byte PicGetIrqState(byte irqLine);
byte isIrqPitEnable(void);

#endif // _PIC_H
```

Αρχείο: proccessStruct.h

```
#ifndef _PROCCESS_STRUCT_H
#define _PROCCESS_STRUCT_H

#include "type.h"

#define MAX_NUM_PROCCES 256

#define PROCCES_VER 2
```

```
#if  PROCCES_VER == 2
typedef struct SpageFrameRegionEntry{
   void *base;
   size_t length;
   dword type;
   dword reserved; /*to keep struct page aligned*/
} pageFrameRegionEntry;

typedef struct SpageFrameCatalog{
   size_t count; /*number of entries in catalog*/
   pageFrameRegionEntry *catalog;
} pageFrameCatalog;

typedef struct SchildProccesNode{
   dword pid;
   int retVal;
   bool hasReturn;
   struct SpcbNode *pcb;
   struct SchildProccesNode *next;
} childProccesNode;

typedef enum{
   ready = 0x0,
   blocked = 0x1
} proccesState;

#define PATH_MAX_LENGTH       128
#define MAX_OPEN_FILES        16
#define INVLD_FILE_CUR_POS    0xFFFFFFFF//(0xF << sizeof(size_t) )
#define INVLD_FILE_INDX       0xFFFFFFFF
#define PATH_NOT_FOUND        0xFFFFFFFE
```

```
typedef struct Sfile{
   dword fileHndlr;
   char fpath[PATH_MAX_LENGTH];
   size_t pos;
   size_t fsize;
   dword fsIndx;
} file;

typedef struct Tcond_t{
   //pcbNode *proc;
   dword code;
} cond_t;



#include "sysCallStructs.h"  /***proccesMsg struct*/

#define MSG_QUEUE_SIZE      0x40

typedef struct SmessageQueue{
   proccesMsg procMsg[MSG_QUEUE_SIZE];
   dword first, last;
   cond_t cond_emtpyQueue;
} messageQueue;

#define  P_TYPE_BACKGROUND  0x00
#define  P_TYPE_GUI         0x01

typedef struct SpcbNode{
   struct SpcbNode *next, *parentPcb;
   dword pid, parentPid, pType;
   childProccesNode *childPidListHead;
   register32 esp, ebp, esp0;
```

```
    pageFrameCatalog pageCat;
    proccesState state;
    file openfiles[MAX_OPEN_FILES];
    messageQueue msgQueue;
    byte *videoBuffer;

} pcbNode;

#endif // defined

#endif // _PROCCESS_STRUCT_H
```

Αρχείο: ProcCondition.h

```
#ifndef _PROC_CONDITION_H
#define _PROC_CONDITION_H

#include "type.h"
#include "proccessStruct.h"

void init_condition(cond_t *cond);
void cond_add_proc(cond_t *cond, pcbNode *curr);
```

```
#endif // _PROC_CONDITION_H
```

Αρχείο: procScheduler.h

```
#ifndef _PROCSCHEDULER_H
#define _PROSCSHEDULER_H

#include "type.h"
#include "memory.h"

#define SUCCED 0x0
#define FAIL_MAX_NUM_OF_PROCCES 0x1
```

```
void setCurrEsp0(dword);

void pit_interrupt_handler(void);///defined in file pit_interupt.asm
//dword newProcces(address);
void exitPorcces(int);
void showStack(void);
void selectProcces(void);
void initProc(void);
int wait(int*);

#endif // _PROCSCHEDULER_H
```

Αρχείο: registers.h

```
#ifndef _REGISTERS_H
#define _REGISTERS_H

#include "type.h"


#define SET_AX(a) __asm volatile("movw %0, %%ax;"   \
                        :               \
                        : "r" ((word) a ) )
```

```
#define SET_DS(a) __asm volatile("movw %0, %%ds;"   \
                        :               \
                        : "a" ((word) a ) )

#define SET_ES(a) __asm volatile("movw %0, %%es;"   \
                        :               \
                        : "a" ((word) a ) )

#define SET_FS(a) __asm volatile("movw %0, %%fs;"   \
                        :               \
                        : "a" ((word) a ) )

#define SET_GS(a) __asm volatile("movw %0, %%gs;"   \
                        :               \
                        : "a" ((word) a ) )

#define SET_SS(a) __asm volatile("movw %0, %%ss;"   \
                        :               \
                        : "a" ((word) a ) )

#define GET_EAX(a) __asm volatile("movl %%eax, %0;"    \
                        : "=r" ((dword)(a))   \
                        :             )

#define SET_EAX(a) __asm volatile("movl %0, %%eax;"   \
                        :               \
                        : "r" ((dword) a ) )

#define SET_EBX(a) __asm volatile("movl %0, %%ebx;"   \
                        :               \
                        : "r" ((dword) a ) )
```

```
#define SET_ECX(a) __asm volatile("movl %0, %%ecx;"   \
                        :               \
                        : "r" ((dword) a ) )

#define SET_EBP(a) __asm volatile("movl %0, %%ebp;"   \
                        :               \
                        : "r" ((dword) (a) ) )

#define SET_ESP(a) __asm volatile("movl %0, %%esp;"   \
                        :               \
                        : "r" ((dword) (a) ) )

#define GET_ECX(a) __asm volatile("movl %%ecx, %0;"    \
                        : "=r" ((dword)(a))   \
                        :           )

#define GET_EDX(a) __asm volatile("movl %%edx, %0;"    \
                        : "=r" ((dword)(a))   \
                        :           )

#define GET_EBX(a) __asm volatile("movl %%ebx, %0;"    \
                        : "=r" ((dword)(a))   \
                        :           )

#define GET_EBP(a) __asm volatile("movl %%ebp, %0;"    \
                        : "=r" ((dword)(a))   \
                        :           )

#define GET_ESI(a) __asm volatile("movl %%esi, %0;"    \
                        : "=r" ((dword)(a))   \
                        :           )
```

```
#define GET_EDI(a) __asm volatile("movl %%edi, %0;"    \
                        : "=r" ((dword)(a))   \
                        :             )

#define GET_ESP(a) __asm volatile("movl %%esp, %0;"    \
                        : "=r" ((dword)(a))   \
                        :             )


#endif /// _REGISTERS_H
```

Αρχείο: segments.h

```
#ifndef _SEGMENTS_H
#define _SEGMENTS_H

#define CS_PL0      0x08
#define DS_PL0      0x10

#define CS_PL3      0x1B
#define DS_PL3      0x23
```

```
#define TSS_SEG     0x2B

#endif // _SEGMENTS_H
```

Αρχείο: stdarg.h

```
#ifndef _STDARG_H
#define _STDARG_H

#ifndef  _VA_LIST
#define _VA_LIST

typedef char* va_list;

#endif
```

```
/*
 * Amount of space required in an argument list (ie. the stack) for an
 * argument of type t.
 */
#define __va_argsiz(t)    \
        (((sizeof(t) + sizeof(int) - 1) / sizeof(int)) * sizeof(int))


#define va_start(ap, pN)            \
        ((ap) = ((va_list) (&pN) + __va_argsiz(pN)))


#define va_end(ap)       ((void)0)
#define va_arg(ap, t)                                           \
         (((ap) = (ap) + __va_argsiz(t)),               \
          *((t*) (void*) ((ap) - __va_argsiz(t))))


#endif // _STDARG_H
```

Αρχείο: stdio.h

```
#ifndef _STDIO_H
#define _STDIO_H

#include "stdarg.h"
#include "type.h"

#define OFFSET(strct, member)   ( (dword) ((void *)(&(member)) - (void *)(&(strct))) )

int printf(char *ch, ... );
```

```
void memcpy(byte *source, byte *dest, size_t count);
void memset(void* buffer, int val, size_t count);

#endif // _STDIO_H
```

Αρχείο: stdiok.h

```
#ifndef _STDIOK_H
#define _STDIOK_H

int printk(char *ch, ... );

#endif // _STDIOK_H
```

Αρχείο: string.h

```
#ifndef _STRING_H
#define _STRING_H

#include "type.h"

size_t strlen(const char* str);
char *strncpy(char *destination, const char *source, size_t num);
char *strncat(char *destination, const char *source, size_t num);
char *strcpy(char *destination, const char *source);
```

```
char *strcat(char *destination, const char *source);
int strncmp(const char *str1, const char *str2, size_t num);
int strcmp(const char *str1, const char *str2);
char *inttostr(size_t val, size_t base, char *destination);
bool isLetter(const char);
size_t copyBytes(byte *destination, const byte *source, size_t num);
size_t strcspn(const char *str1, const char *str2);

#endif // _STRING_H
```

Αρχείο: sysCallLib.h

```
#ifndef _SYSCALLLIB_H
#define _SYSCALLLIB_H

#include "type.h"

typedef struct TfileAttr{
   char fName[128];
} fileAttr;
```

```
void ScPrintChar(char charToPrint, byte color);
void* allocatePhysicalPages(size_t );
int freePhysicalPages(void* );
dword createProcces(const char *exepath);
void exitProc(int );
int waitProc(int*);
dword readFile(dword filehndlr, size_t count, byte* buffer);
dword writeFile(dword fileHndlr, size_t count, byte* buffer, int flags);
dword openFile(const char *path, const char *rw);
dword seekFile(dword filehndlr, long int offset, int origin);
void closeFile(dword filehndlr);
dword getMessage(void* msg);
dword setProcProperties(dword procDescr);
dword setActiveProc(dword procHndl);
void draw(byte*, size_t, size_t);
dword fileExists(const char*path);
dword getFileAttr(const char* fpath, const dword fromFile, size_t countFiles,  fileAttr *fileAtt);

#define ScPrintChar_NR              0
#define freePhysicalPages_NR        1
#define allocatePhysicalPages_NR    2
#define createProcces_NR            3
#define exitProc_NR                 4
#define waitProc_NR                 5
#define readFile_NR                 6
#define writeFile_NR                7
#define openFile_NR                 8
#define seekFile_NR                 9
#define closeFile_NR                10
#define getMessage_NR               11
#define setProcProperties_NR        12
#define setActiveProc_NR            13
```

```
#define draw_NR                 14
#define fileExists_NR           15
#define getFileAttr_NR          16

/*From https://www.win.tue.nl/~aeb/linux/lk/lk-4.html*/
#define makeSysCall0(type, name)        \
type name(void){                        \
   size_t res;                          \
   __asm volatile("int $255;"           \
            :"=a" (res)                 \
            :"0" (name##_NR)            \
            :);                         \
   return (type) res;                   \
}


#define makeSysCall1(type, name, type0, arg0)       \
type name(type0 arg0){                              \
   size_t res;                                      \
   __asm volatile("int $255;"                       \
            :"=a" (res)                             \
            :"0" (name##_NR), "b" (arg0)            \
            :);                                     \
   return (type) res;                               \
}

#define makeSysCall2(type, name, type0, arg0, type1, arg1)    \
type name(type0 arg0, type1 arg1){                            \
   size_t res;                                                \
   __asm volatile("int $255;"                                 \
            :"=a" (res)                                       \
            :"0" (name##_NR), "b" (arg0), "c" (arg1)          \
```

```
        :);                                 \
    return (type) res;                      \
}

#define makeSysCall3(type, name, type0, arg0, type1, arg1, type2, arg2)     \
type name(type0 arg0, type1 arg1, type2 arg2){                              \
    size_t res;                                                             \
    __asm volatile("int $255;"                                              \
        :"=a" (res)                                                         \
        :"0" (name##_NR), "b" (arg0), "c" (arg1), "d"(arg2)                 \
        :);                                                                 \
    return (type) res;                                                      \
}

#define makeSysCall4(type, name, type0, arg0, type1, arg1, type2, arg2, type3, arg3)     \
type name(type0 arg0, type1 arg1, type2 arg2, type3 arg3){                               \
    size_t res;                                                                          \
    __asm volatile("int $255;"                                                           \
        :"=a" (res)                                                                      \
        :"0" (name##_NR), "b" (arg0), "c" (arg1), "d"(arg2), "S"(arg3)                   \
        :);                                                                              \
    return (type) res;                                                                   \
}

#endif // _sysCallLib_H
```

Αρχείο: sysCallStructs.h

```
#ifndef _SYSCALLSTRUCTS_H
#define _SYSCALLSTRUCTS_H

#include "type.h"

typedef struct SproccesMsg{
   dword msgType;
   dword msg;
   dword param;
```

```
} proccesMsg;

#endif // _SYSCALLSTRUCTS_H
```

Αρχείο: sysCallTable.h

```
#ifndef _SYSCALLTABLE_H
#define _SYSCALLTABLE_H

#include "type.h"
#include "vFileOperations.h"

extern void printChar(char, byte);
extern int ScFreePhysicalPages(void* );
extern void* ScMemAlloc(size_t, dword, dword, dword);
```

```
extern dword SccreateProcces(const char *exepath, const char * params);

extern void ScdestroyProcces(int);   //exitPorcces(int );

extern int wait(int*);

extern void ScReadFile(void*, size_t, byte*);

extern void ScWriteFile(void*, size_t, byte*, int);

extern dword Scfopen(const char*, const char*);

extern dword ScseekFile(dword, long int, int );

extern void ScccloseFile(dword filehndlr);

extern dword ScgetMessage(proccesMsg *msg);

extern dword ScsetProcProperties(dword procDescr);

extern dword ScsetActiveProc(dword procHndl);

extern void ScDraw(byte*, size_t, size_t, dword);

extern dword ScfileExists(const char*path, dword *attr, size_t fsize);

extern dword ScGetFileAttr(const char* fpath, const dword fromFile, size_t countFiles,  fileAttr
*fileAtt);

extern dword Scfcreate(const char *path);

extern dword ScLoadDriver(const char *driverpath);



dword sysCallTable[] = {(dword) printChar, (dword) ScFreePhysicalPages, (dword) ScMemAlloc,
(dword) SccreateProcces,

        (dword) ScdestroyProcces, (dword) wait, (dword) ScReadFile, (dword) ScWriteFile,
(dword) Scfopen, (dword) ScseekFile,

        (dword) ScccloseFile, (dword) ScgetMessage, (dword) ScsetProcProperties, (dword)
ScsetActiveProc, (dword) ScDraw,

        (dword) ScfileExists, (dword) ScGetFileAttr, (dword) Scfcreate, (dword)
ScLoadDriver};


#endif // _SYSCALLTABLE_H
```

Αρχείο: type.h

```
#ifndef _TYPE_H
#define _TYPE_H

#define NULL    (void*)0


typedef char    s8bit;
typedef unsigned char   u8bit;
typedef unsigned char   byte;
```

```
typedef enum {
  false = 0,
  true = 1
} bool;

typedef short int   s16bit;
typedef unsigned short int  u16bit;
typedef unsigned short int  word;

#ifdef __x86_64__
#error 64bit code is not supported!!
#else
#define __WORDSIZE 32
#endif // __x86_64__

#if __WORDSIZE == 32
typedef int s32bit;
typedef unsigned int u32bit;
typedef unsigned int dword;
typedef unsigned int size_t;
#endif // _WORDSIZE

typedef long long qword;   /**??unsigned*/

typedef long long long64;

typedef dword   register32;
typedef word    register16;

typedef register32 address;

typedef int int32_t;
```

```
typedef unsigned int uint32_t;

typedef long long int64_t;
typedef unsigned long long uint64_t;

#endif /// _TYPE_H
```

Αρχείο: version.h

```
#ifndef _VERSION_H
#define _VERSION_H

#ifndef __GNUC__
#error compiler not supported
#endif // gcc compiler

#if __GNUC__ < 7 || \
   (__GNUC__ == 7 && (__GNUC_MINOR__ < 2 ))
#error gcc version < 7.2.0
```

```
#endif // compile if version >= 7.2.0

#endif // _VERSION_H
```

Αρχείο: vfileOperations.h

```
#ifndef V_FILE_OPERATIONS
#define V_FILE_OPERATIONS


typedef struct TfileAttr{
   char fName[128];
} fileAttr;

typedef struct Soperations{
   size_t (*read)(dword, size_t, void*);
```

```
   size_t (*write)(dword, size_t, void*, int);
   dword (*fopen) (const char*, const char*);
   dword (*fseek) (dword, long64, size_t );
   dword (*getSize)(void);
   //dword (*setSize)(size_t fsize, dword fileHandle);
   dword (*fcreate) (const char*);
   void (*fclose) (dword);
   dword (*fexists) (const char*, dword *attr, size_t *fsize);
   dword (*list_dir) (const char* fpath, const dword fromFile, size_t countFiles,  fileAttr *fileAtt);

} operations;

#endif // V_FILE_OPERATIONS
```

Αρχείο: vfs.h

```
#ifndef _VFS
#define _VFS

#include "type.h"

#include "vfileOperations.h"

/**Scfseek */
#define SC_FSSEK_FAILED     0x0000FFFF
#define SEEK_SET    0x10
```

```
#define SEEK_CUR    0x20
#define SEEK_END    0x30

dword ScReadFile(dword, size_t, byte*);
dword ScWriteFile(dword, size_t, byte*, int);
dword Scfopen(const char *path, const char * rw);
dword vfread(const char *path, dword *fsIndx, size_t offset, size_t count, void *buffer);
dword vfwrite(const char *path, size_t offset, size_t count, void *buffer, int flags);
dword ScseekFile(dword fileHndlr, long int offset, int origin );
dword ScfileExists(const char*path, dword *attr, size_t *fsize);
dword ScGetFileAttr(const char* fpath, const dword fromFile, size_t countFiles,  fileAttr *fileAtt);
dword Scfcreate(const char *path);

/**for use without proccess**/
dword vfopen(const char *path, const char * rw, size_t * fsize);

#define STORAGE_DEV_STRING  "storage"


#endif // _VFS
```

Αρχείο: vga.h

```
#ifndef _VGA_H
#define _VGA_H

#include "type.h"

#define ADDRESS_ATTRIBUTE_REGISTER      0x3C0
#define DATA_WRITE_ATTRIBUTE_REGISTER   0x3C0
#define DATA_READ_ATTRIBUTE_REGISTER    0x3C1
#define CRTC_ADDRESS_REGISTER           0x3D4
#define CRTC_DATA_REGISTER              0x3D5
```

```
#define INPUT_STATUS_1_REGISTER        0x3DA

///Indexes for CRTC registers
#define CURSOR_LOCATION_LOW     0x0F
#define CURSOR_LOCATION_HIGH    0x0E
#define CURSOR_START_REGISTER   0x0A

///Indexes for Attribute registers
#define MODE_CONTROL_REG_INX    0x10

///Vga modes(values in mode control register)
#define TEXT_MODE_80x25            0x0C
#define PLANAR_640x480_MODE        0x01
#define LINEAR_320x200_MODE        0x41

typedef struct struct_cursor_position{
   word row;
   word column;
} cPosition;


void printChar(char charToPrint, byte color);
void clearScreen();
void disableCursor(void);
void enableCursor(void);
void setCursorPosition(word row, word column);
void getCursorPosition(cPosition *pos);
void printString(char *string, byte color);
void init_vga(void);
void vga_write(byte *buffer);

typedef struct Svga_state{
```

```
   dword vga_mode;
}vga_state;

#endif // _VGA_H
```

Αρχείο: volume.h

```
#ifndef _VOLUME_H
#define _VOLUME_H

#include "type.h"
#include "device.h"

#define MAX_MBR_ENTRIES 4

#define MAX_VOLUMES_COUNT      8  /**max_volume_supported is better*/
```

```
#define MAX_VPATH_LENGTH    3 /*Letter + ':' + null*/

/*This constants are MBR specific*/
#define HPC           255
#define SPT           63
#define LBA_TO_CHS(lba, c, h, s)       ((c) = ( (lba) / (SPT * HPC) )),    \
                              ((h) = ( ( (lba) / SPT ) % HPC ) ), \
                              ((s) = ( ( (lba) % SPT) + 1 ) )
#define CHS_TO_LBA(lba, c, h, s)       ((lba) = ( ((c) * (SPT * HPC)) + ((h) * SPT) + ((s) - 1 ) ) )
#define CHS_MAX_SECTORS     1024 * 255 * 63   /*Max count of sectors that can be represented with CHS*/

#define PARTITION_ENTRY_SIZE       0x10
#define PARTITION_1_ENTRY_OFFSET    0x1BE
#define PARTITION_2_ENTRY_OFFSET    0x1CE
#define PARTITION_3_ENTRY_OFFSET    0x1DE
#define PARTITION_4_ENTRY_OFFSET    0x1EE

#define PARTITION_IS_BOOTABLE_OFFSET       0x00
#define PARTITION_IS_BOOTABLE              0x80
#define PARTITION_NOT_BOOTABLE             0x00

#define START_SECTOR_CHS_OFFSET         0x01
#define PARTITION_TYPE_OFFSET           0x04
#define LAST_SECTOR_CHS_OFFSET          0x05
#define START_SECTOR_LBA_OFFSET         0x08
#define TOTAL_SECTORS_LBA_OFFSET        0x0C

#define MBR_SIGNATURE_OFFSET       0x1FE
#define MBR_SIGNATURE              0xAA55 /*Little endian*/

typedef struct SmbrEntry{
   byte isBootable;
```

```
   byte chsStartSector, chsEndSector;
   byte chsStartHead, chsEndHead;
   word chsStartCylinder, chsEndCylinder;
   byte partitionType;
   dword lbaStart;
   dword lbaLength;
} mbrEntry;

typedef struct Svolume{
   char vpath[MAX_VPATH_LENGTH];
   char deviceId[MAX_ID_STRING]; /*to confirm write at the correct disk*/
   dword lbaStart;
   dword partitionSize;
   size_t fs_indx;
   //device id /*device id*/
} volume;

void partitionCreate(const char *disk, dword lbaStart, size_t length);
dword VolumeRead(const char *path, size_t offset, size_t count, byte *buffer);
dword VolumeWrite(const char *path, size_t offset, size_t count, byte *buffer);
void initVolumes(void);
int volumeSearch(const char *);
dword getFsIndx(const char *);

extern volume volumes[];

#endif // _VOLUME_H
```

Αρχείο: waitProc.h

```
#ifndef _WAIT_PROC_H
#define _WAIT_PROC_H

#include "type.h"

#endif // _WAIT_PROC_H
```

Προγράμματα και βιλβλιοθήκες στον χώρο του χρήστη:

C Library:

Αρχείο: ctype.c

```
#include "ctype.h"

#define TRUE    0x01
#define FALSE   0x00

int isdigit(int c)
{
```

```
   return ( (c >= 0x30) && (c <= 0x39) ) ? TRUE : FALSE;
}

int isprint (int c)
{
   return c > 0x1F  &&  c != 0x7F;
}

int isgraph(int c)
{
   return ( c > 0x1F  &&  c != 0x7F ) || c == ' ';
}

int isspace(int c)
{
   return c == ' '  || c == '\t' ||  c == '\v'  ||  c == '\n' ||  c == '\f'  ||  c == '\r';
}
```

Αρχείο: ctype.h

```
#ifndef _CTYPE_H
#define _CTYPE_H

#include "type.h"

#ifdef __cplusplus
 extern "C"{
#endif // __cplusplus

int isprint (int c);
int isgraph (int c);
```

```
int isdigit(int c);
int isspace(int c);



#ifdef __cplusplus
  }
#endif // __cplusplus

#endif // _CTYPE_H
```

Αρχείο: fileRead.cc

```
#include "fileRead.hpp"
#include "sysCallLib.h"   ///?? sysCall better name

dword (*stdRead)(size_t, byte*);

bool setStdRead(dword (f)(size_t, byte*))
{
   stdRead = f;
   return true;
}
```

```
dword fileRead(dword filehndlr, size_t count, byte* buffer)
{
   if( filehndlr == STDIN )
      return stdRead(count, buffer);

   return readFile(filehndlr, count, buffer);
}
```

Αρχείο: fileRead.hpp

```
#ifndef _FILE_READ_H
#define _FILE_READ_H

#include "type.h"

#define STDIN   0xFFFFFFF0
#define STDOUT  0xFFFFFFF1

#ifdef __cplusplus
 extern "C"{
#endif // __cplusplus
```

```
extern dword (*stdRead)(size_t, byte*);

bool setStdRead(dword (f)(size_t, byte*));
dword fileRead(dword filehndlr, size_t count, byte* buffer);

#ifdef __cplusplus
 }
#endif // __cplusplus

#endif // _FILE_READ_H
```

Αρχείο: fileWrite.cc

```
#include "fileWrite.hpp"
#include "sysCallLib.h"

dword (*stdWrite)(size_t, byte*);

bool setStdWrite(dword (f)(size_t, byte*))
{
   stdWrite = f;
   return true;
}
```

```
dword fileWrite(dword filehndlr, size_t count, byte* buffer, int flags)
{
   if( filehndlr == STDOUT )
      return stdWrite(count, buffer);

   return writeFile(filehndlr, count, buffer, flags);
}
```

Αρχείο: fileWrite.hpp

```
#ifndef _FILE_WRITE_H
#define _FILE_WRITE_H

#include "type.h"

#define STDIN   0xFFFFFFF0
#define STDOUT  0xFFFFFFF1

#ifdef __cplusplus
 extern "C"{
#endif // __cplusplus
```

```
extern dword (*stdWrite)(size_t, byte*);

bool setStdWrite(dword (f)(size_t, byte*));
dword fileWrite(dword filehndlr, size_t count, byte* buffer, int flags);

#ifdef __cplusplus
  }
#endif // __cplusplus

#endif // _FILE_WRITE_H
```

Αρχείο: osUtils.c

```
#include "osUtils.h"
#include "string.h"

char* pathToAbs(char *relPath, const char *absPath)
{
   size_t len = strlen(relPath);

   if(len > 0)
      if( relPath[len - 1] != '/' )
         strcat(relPath, "/");
```

```
   strcat(relPath, absPath);

   return relPath;
}

bool pathIsAbsolute(const char *path)
{

   if( isLetter(path[0])  && path[1]  == ':' )
      return true;

   return false;
}

void normalizePaths(char *result, const char *relPath, const char *absPath)
{
   strcpy(result, absPath);

   //if(!strlen(relPath))
      //strcpy(relPath, absPath);

   if( !pathIsAbsolute(relPath) )
      pathToAbs(result, relPath);
   else
      strcpy(result, relPath);

   removeGoToParentFromPath(result+3);
}

/*
int getNDirPos(const char *path, size_t dirIndx)
{
```

```
}
*/

static char* getParentDir(const char *path, char *curr)
{
   if(curr - 2 >= path)
      curr -= 2;

   while(curr > path  && *curr != '/')
      curr--;

   if(*curr == '/')
      curr++;

   return curr;
}

/// A:/ A1/../A2   /// SET/A1/../A2
char* removeGoToParentFromPath(char *path)
{
   char *curr, *ptr = NULL;
   size_t pDirStrLen = strlen(PARENT_DIR_STR);

   curr = path;
   while(curr != NULL)
   {

      if( !strncmp(curr, PARENT_DIR_STR, pDirStrLen )  ){
         ptr = getParentDir(path, curr);
         memmove( ptr, curr + pDirStrLen, strlen(curr + pDirStrLen) + 1 );
         curr = ptr;
```

```
        }
        else{
            curr = strchr(curr,'/');

            if(curr != NULL)
                curr++;
        }

    }

    return path;
}
```

Αρχείο: osUtils.h

```
#ifndef _OS_UTILS_H
#define _OS_UTILS_H

#include "type.h"

#define PARENT_DIR_STR     "../"
#define MAX_PATH_LEN       256   ///??Param from file

#ifdef __cplusplus
 extern "C"{
#endif // __cplusplus
```

```
char* pathToAbs(char *relPath, const char *absPath);
bool pathIsAbsolute(const char *path);
char* removeGoToParentFromPath(char *path);
void normalizePaths(char *result, const char *relPath, const char *absPath);


#ifdef __cplusplus
 }
#endif // __cplusplus

#endif /// _OS_UTILS_H
```

Αρχείο: registers.h

```
#ifndef _REGISTERS_H
#define _REGISTERS_H

#include "type.h"


#define SET_AX(a) __asm volatile("movw %0, %%ax;"   \
                        :               \
                        : "r" ((word) a ) )

#define SET_DS(a) __asm volatile("movw %0, %%ds;"   \
```

```
                        :               \
                        : "a" ((word) a ) )

#define SET_ES(a) __asm volatile("movw %0, %%es;"   \
                        :               \
                        : "a" ((word) a ) )

#define SET_FS(a) __asm volatile("movw %0, %%fs;"   \
                        :               \
                        : "a" ((word) a ) )

#define SET_GS(a) __asm volatile("movw %0, %%gs;"   \
                        :               \
                        : "a" ((word) a ) )

#define SET_SS(a) __asm volatile("movw %0, %%ss;"   \
                        :               \
                        : "a" ((word) a ) )

#define GET_EAX(a) __asm volatile("movl %%eax, %0;"    \
                        : "=r" ((dword)(a))   \
                        :           )

#define SET_EAX(a) __asm volatile("movl %0, %%eax;"   \
                        :               \
                        : "r" ((dword) a ) )

#define SET_EBX(a) __asm volatile("movl %0, %%ebx;"   \
                        :               \
                        : "r" ((dword) a ) )

#define SET_ECX(a) __asm volatile("movl %0, %%ecx;"   \
```

```
                    :               \
                    : "r" ((dword) a ) )

#define SET_EBP(a) __asm volatile("movl %0, %%ebp;"   \
                    :               \
                    : "r" ((dword) (a) ) )

#define SET_ESP(a) __asm volatile("movl %0, %%esp;"   \
                    :               \
                    : "r" ((dword) (a) ) )

#define GET_ECX(a) __asm volatile("movl %%ecx, %0;"    \
                    : "=r" ((dword)(a))   \
                    :           )

#define GET_EDX(a) __asm volatile("movl %%edx, %0;"    \
                    : "=r" ((dword)(a))   \
                    :           )

#define GET_EBX(a) __asm volatile("movl %%ebx, %0;"    \
                    : "=r" ((dword)(a))   \
                    :           )

#define GET_EBP(a) __asm volatile("movl %%ebp, %0;"    \
                    : "=r" ((dword)(a))   \
                    :           )

#define GET_ESI(a) __asm volatile("movl %%esi, %0;"    \
                    : "=r" ((dword)(a))   \
                    :           )

#define GET_EDI(a) __asm volatile("movl %%edi, %0;"    \
```

```
                : "=r" ((dword)(a))   \
                :             )

#define GET_ESP(a) __asm volatile("movl %%esp, %0;"    \
                : "=r" ((dword)(a))   \
                :             )


#endif /// _REGISTERS_H
```

Αρχείο: stdarg.h

```
#ifndef _STDARG_H
#define _STDARG_H

#ifndef  _VA_LIST
#define _VA_LIST

typedef char* va_list;

#endif
```

```
/*
 * Amount of space required in an argument list (ie. the stack) for an
 * argument of type t.
 */
#define __va_argsiz(t)    \
        (((sizeof(t) + sizeof(int) - 1) / sizeof(int)) * sizeof(int))


#define va_start(ap, pN)            \
        ((ap) = ((va_list) (&pN) + __va_argsiz(pN)))


#define va_end(ap)       ((void)0)

#define va_arg(ap, t)                                           \
         (((ap) = (ap) + __va_argsiz(t)),               \
          *((t*) (void*) ((ap) - __va_argsiz(t))))

#endif // _STDARG_H
```

Αρχείο: stdio.c

```
#include "stdio.h"
#include "type.h"
#include "sysCallLib.h"
//#include "sysCallStructs.h"
#include "ctype.h"
#include "stdlib.h"
#include "fileRead.hpp"
#include "fileWrite.hpp"

/*heapNode *memListHead;
```

```
static void* createReservedNode(heapNode *curr, size_t allocSize);
static heapNode* findFreeNode(size_t allocSize);
heapNode* findReservedRegion(void* ptr, heapNode** prevFree);
heapNode* mergeIfNeighbour(heapNode *prev, heapNode *curr);
*/


byte emptyDraw[80*25];
size_t temp_vidIndx;

void stdInit(void)
{

   temp_vidIndx = 0;
   memset(emptyDraw, ' ', 80*25);

   initKernelHeap((void*) 0x005AF000,  1024*1024);     ///?? First param not used/need
}
/*
static size_t vid_inc(size_t *t)
{


   size_t res = *t;

   if(*t < 80*25-1)
      (*t)++;
   else{
      *t = 0;
      draw(emptyDraw, 0, 80*25, 0);
   }
```

```
   return res;
}*/


int printf(const char *ch, ...)     /**??printf must call sprintf*/
{
   va_list arg;
   dword val, i, j = 28;
   char symbolTable[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
   //size_t pad;


   va_start(arg, ch);

   if(ch == NULL)
      return -1;

   while(*ch != '\0')
   {
      switch(*ch){
         case '%':
            ch++;
            switch( *ch ){
               case '%':
                  //ScPrintChar(*ch, 0x0F);
                  fileWrite(STDOUT, 1, (byte*) ch, 0 );
                  break;
               case 'x':
                  val =(dword) va_arg(arg, dword);
                  for(i =(dword) 0xF0000000; i >= (dword)0xF; i >>= 4, j-=4 )
```

```
                    fileWrite(STDOUT, 1, (byte*) &symbolTable[ (val & i) >> j ], 0 );
//draw((byte*)&symbolTable[ (val & i) >> j ], vid_inc(&temp_vidIndx), 1,
0);//ScPrintChar(symbolTable[ (val & i) >> j ], 0x0F);
                    break;
                case 'c':
                    val =(byte) va_arg(arg, byte);
                    fileWrite(STDOUT, 1, (byte*) &val, 0 ); //draw((byte*)&val,
vid_inc(&temp_vidIndx), 1, 0);  //ScPrintChar(val, 0x0F);
                    break;
                case 's':
                    val = (dword) va_arg(arg, dword);
                    printf( (char*) val);   ///?? if %s char exist undifned behavor
                    break;
            }

        break;
        /*case '\n':

            if(temp_vidIndx > 24*80){
                draw(emptyDraw, 0, 80*25, 0);
                temp_vidIndx = 0;
            }
            else{
                pad = 80-(temp_vidIndx % 80);
                draw(emptyDraw, temp_vidIndx, pad, 0 );
                temp_vidIndx += pad;
            }

            break;*/

        default:
            fileWrite(STDOUT, 1, (byte*) ch, 0 );  //draw((byte*)ch, vid_inc(&temp_vidIndx), 1, 0);
//ScPrintChar(*ch, 0x0F);
            break;
```

```
        }

        ch++;
    }

    va_end(arg);

    return 0;
}

int sprintf(char *str, const char *ch, ...)
{
    va_list arg;
    dword val, i, j = 28;
    char symbolTable[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
    size_t k = 0;


    va_start(arg, ch);

    if(ch == NULL)
        return -1;

    while(*ch != '\0')
    {
        switch(*ch){
            case '%':
                ch++;
                switch( *ch ){
                    case '%':
```

```
                str[k++] = *ch;  //fileWrite(STDOUT, 1, (byte*) ch, 0 );
                break;
            case 'x':
                val =(dword) va_arg(arg, dword);
                for(i =(dword) 0xF0000000; i >= (dword)0xF; i >>= 4, j-=4 )
                   str[k++] = symbolTable[ (val & i) >> j ];   //fileWrite(STDOUT, 1, (byte*)
&symbolTable[ (val & i) >> j ], 0 );
                break;
            case 'c':
                val =(byte) va_arg(arg, byte);
                str[k++] = val;   //fileWrite(STDOUT, 1, (byte*) &val, 0 );
                break;
            case 's':
                val = (dword) va_arg(arg, dword);
                sprintf(str+k, (char*) val);
                break;
          }

        break;


        default:
          str[k++] = *ch;    //fileWrite(STDOUT, 1, (byte*) ch, 0 );
          break;


    }

    ch++;
  }

  va_end(arg);
```

```
   str[k] = '\0';
   return k;
}

int atoi(const char *str)
{
   int num = 0;

   while( isdigit(*str) )
      num = num * 10 + (*str++ - 0x30 );

   return num;
}

char getChar(void)
{
   proccesMsg msg;

   getMessage(&msg);
   return (char) msg.msg;
}

char *gets (char *str)
{
   //proccesMsg msg;
   //char *p = str;
   //char ch;


   fileRead(STDIN, 100000, (byte*) str);
```

```
   /*getMessage(&msg);
   while( msg.msg != '\n' || msg.msgType != 0x01)
   {
      ch = msg.msg;

      if( isgraph( ch )  &&  msg.msgType == 0x01 )
      {
         *p++ = ch;
         printf("%c", ch);  //ScPrintChar(ch, 0x0F);
      }

      getMessage(&msg);
   }

   printf("\n");
   *p = '\0';*/

   return str;
}

void memset(void* buffer, int val, size_t count)
{
   char *p = buffer;

   while(count--)
      *p++ = (unsigned char)val;
}
```

Αρχείο: stdio.h

```
#ifndef _STDIO_H
#define _STDIO_H

#include "stdarg.h"
#include "type.h"

#define OFFSET(strct, member)   ( (dword) ((void *)(&(member)) - (void *)(&(strct))) )

#ifdef __cplusplus
 extern "C"{
```

```
#endif // __cplusplus

int printf(const char *ch, ... );
int sprintf(char *str, const char *ch, ...);
int atoi(const char *str);
char *gets (char *str);
//void memcpy(byte *source, byte *dest, size_t count);
void memset(void* buffer, int val, size_t count);
char getChar(void);
void stdInit(void);




#ifdef __cplusplus
 }
#endif // __cplusplus

#endif // _STDIO_H
```

Αρχείο: stdlib.c

```
#include "stdlib.h"
#include "type.h"
#include "sysCallLib.h"
//#include "sysCallStructs.h"
#include "ctype.h"

typedef struct SheapNode{
   struct SheapNode *next;
   size_t size;
   bool isFreeSlot;
```

```
} heapNode;


heapNode *memListHead;

static void* createReservedNode(heapNode *curr, size_t allocSize);
static heapNode* findFreeNode(size_t allocSize);
static heapNode* findReservedRegion(void* ptr, heapNode** prevFree);
static heapNode* mergeIfNeighbour(heapNode *prev, heapNode *curr);


int initKernelHeap(void *heap_base, size_t heapSz)
{

   heap_base = memAlloc(heapSz / PAGE_FRAME_SIZE, (dword) heap_base /
PAGE_FRAME_SIZE, ALLOC_MODE_ANY_ADD, 0);


   if(heap_base == NULL)
   {
      memListHead = (heapNode*) NULL;
      return 0;
   }

   memListHead = (heapNode*) heap_base;
   memListHead->next = NULL;
   memListHead->isFreeSlot = true;
   memListHead->size = heapSz;

   return 1;

}
```

```
void *malloc(size_t allocSize)
{
   heapNode *curr = NULL;

   if((curr = findFreeNode(allocSize)) == NULL)
      return NULL;

   return createReservedNode(curr, allocSize);
}

static heapNode* findFreeNode(size_t allocSize)
{
   heapNode *curr = memListHead, *prev = NULL, *tmp;

   while(curr != NULL)
   {
      if( (curr->size >= allocSize + sizeof(heapNode) ) && (curr->isFreeSlot == true) )
         return curr;

      prev = curr;
      curr = curr->next;
   }

   ///tryExpand
   if(prev != NULL)
   {
      tmp =  memAlloc(allocSize  / PAGE_FRAME_SIZE + 2, (dword) ((size_t) prev + prev->size +
sizeof(heapNode)) / PAGE_FRAME_SIZE, ALLOC_MODE_FIXED_ADD, 0);
      if(tmp != NULL)
      {
         prev->next = tmp;
         tmp->next = NULL;
         tmp->isFreeSlot = true;
```

```
            tmp->size = (allocSize  / PAGE_FRAME_SIZE + 2) * PAGE_FRAME_SIZE;
        }
    }

    return NULL;
}

static void* createReservedNode(heapNode *curr, size_t allocSize)
{
    heapNode *newNode;

    if(curr->size - (allocSize + sizeof(heapNode)) > sizeof(heapNode))
    {
        newNode = (heapNode*) ((size_t) curr + allocSize + sizeof(heapNode));
        newNode->next = curr->next;
        curr->next = newNode;

        newNode->size = curr->size - allocSize - sizeof(heapNode);
        curr->size = allocSize;
        curr->isFreeSlot = false;
        newNode->isFreeSlot = true;

        return (void*)((size_t) curr + sizeof(heapNode));
    }
    else if(curr->size > allocSize + sizeof(heapNode))
    {
        curr->isFreeSlot = false;
        return (void*)((size_t) curr + sizeof(heapNode));
    }

    return NULL;
}
```

```
void free(void *ptr)
{
   heapNode *curr = NULL, *prevFreeNode = NULL;

   if( (curr = findReservedRegion(ptr, &prevFreeNode)) == NULL)
      return;

   curr->isFreeSlot = true;

   curr = mergeIfNeighbour(prevFreeNode, curr);
   curr = mergeIfNeighbour(curr, curr->next);

   return;
}

heapNode* mergeIfNeighbour(heapNode *prev, heapNode *curr)
{
   if(prev == NULL)return curr;
   if(curr == NULL)return prev;

   if(prev->next == curr)
   {
      prev->size += curr->size + sizeof(heapNode);
      prev->next = curr->next;

      return prev;
   }
   else
      return curr;
}
```

```
heapNode* findReservedRegion(void* ptr, heapNode** prevFree)
{
   heapNode *curr = memListHead;

   *prevFree = NULL;
   while(curr != NULL)
   {
      if( (ptr == (void*) ((size_t) curr + sizeof(heapNode)) && (curr->isFreeSlot == false)) )
        return curr;

      if(curr->isFreeSlot == true)
         *prevFree = curr;

      curr = curr->next;
   }

   return NULL;
}
```

Αρχείο: stdlib.h

```
#ifndef _STDLIB_H
#define _STDLIB_H

#include "stdarg.h"
#include "type.h"


#ifdef __cplusplus
 extern "C"{
#endif // __cplusplus
```

```
int initKernelHeap(void *heap_base, size_t heapSz);

void *malloc(size_t allocSize);
void free(void *ptr);

#ifdef __cplusplus
 }
#endif // __cplusplus

#endif // _STDLIB_H
```

Αρχείο: string.c

```
#include "string.h"
#include "ctype.h"

size_t strlen(const char* str)
{
   size_t i = 0;

   while(str[i] != '\0')
      i++;

   return i;
```

```
}

char *strncpy(char *destination, const char *source, size_t num)
{
   size_t i = 0;

   while(i < num && source[i] != '\0')
   {
      destination[i] = source[i];
      i++;
   }

   while(i < num)
      destination[i++] = '\0';

   return destination;
}

char *strncat(char *destination, const char *source, size_t num)
{
   size_t i = 0, j = strlen(destination);

   while(i < num  && source[i] != '\0')
      destination[j++] = source[i++];

   destination[j] = '\0';
   return destination;
}

char *strcpy( char *destination, const char *source )
{
   do{
```

```
        *destination++ = *source;
    }while( *source++ != '\0' );

    return destination;
}

char *strcat(char *destination, const char *source)
{
    size_t i = 0, j = strlen(destination);

    do
    {
        destination[j++] = source[i];
    }while(source[i++] != '\0');

    return destination;
}

int strncmp(const char *str1, const char *str2, size_t num)
{
    size_t i = 0;

    while(i < num && (str1[i] != '\0' || str2[i] != '\0') )
    {
        if(str1[i] < str2[i])
            return -1;
        else if(str1[i] > str2[i])
            return 1;

        i++;
    }
```

```
   return 0;
}

int strcmp(const char *str1, const char *str2)
{
   size_t i = 0;

   while( str1[i] == str2[i]  )
   {
      if(str1[i] == '\0')
         return 0;

      i++;
   }

   if(str1[i] < str2[i])
      return -1;
   else
      return 1;

}

size_t strcspn(const char *str1, const char *str2)
{
   size_t cnt, i;

   for(cnt = 0;  cnt < strlen(str1);  cnt++ )
   {

      for(i = 0; i < strlen(str2); i++ )
         if(str1[cnt] == str2[i] )
            return cnt;
```

```
    }

    return cnt;
}

size_t strspn(const char *str1, const char *str2)
{
    return 0;
}

char *strtok( char *str, const char *delimiters )
{
    //static char *lastStr = NULL;

    return NULL;
}

char *strchr (char *str, int character)
{
    while(*str != '\0')
        if( *str == (char) character )
            return str;
        else str++;

    return NULL;
}

void *memmove(void *destination, const void *source, size_t num)
{
    const unsigned char *src = source;
    unsigned char *dst = destination;
    size_t i;
```

```
    if( src > dst)
        for(i = 0;  i < num; i++)
            dst[i] = src[i];
    else if(src < dst)
        for(i = num - 1;  i >= 0; i--)
            dst[i] = src[i];


    return destination;
}

void *memcpy(void *destination, const void *source, size_t num)
{
    return memcpy(destination, source, num);
}

/**This function is not part of the c library*/
char *inttostr(size_t val, size_t base, char *destination)
{
    const int ascii_zero_char_offset = 48;
    char digit = val % base;

    if(base > 10)
        return NULL;

    if(val / base != 0)
        inttostr(val / base, base, destination);

    digit += ascii_zero_char_offset;
    strncat(destination, &digit, 1);
```

```
   return destination;
}

/**This function is not part of the c library*/
bool isLetter(const char ch)
{
   const char ascii_A_offset = 65, ascii_Z_offset = 90, ascii_a_offset = 97, ascii_z_offset = 122;

   return ( (ch >= ascii_A_offset && ch <= ascii_Z_offset) || (ch >= ascii_a_offset && ch <=
ascii_z_offset))? true : false;
}

/**This function is not part of the c library*/
size_t copyBytes(byte *destination, const byte *source, size_t num)
{
   size_t i;

   for(i = 0;  i < num; i++)
      destination[i] = source[i];

   return num;
}

/**This function is not part of the c library*/
/***https://stackoverflow.com/questions/122616/how-do-i-trim-leading-trailing-whitespace-in-
a-standard-way  user Adam Rosenfield Dave Gray***/
char *trimwhitespace(char *str)
{
  char *end;

  // Trim leading space
  while(isspace((unsigned char)*str)) str++;
```

```
 if(*str == 0)  // All spaces?
   return str;

 // Trim trailing space
 end = str + strlen(str) - 1;
 while(end > str && isspace((unsigned char)*end)) end--;

 // Write new null terminator character
 end[1] = '\0';

 return str;
}
```

Αρχείο: string.h

```
#ifndef _STRING_H
#define _STRING_H

#include "type.h"
#ifdef __cplusplus
 extern "C"{
#endif // __cplusplus

size_t strlen(const char* str);
char *strncpy(char *destination, const char *source, size_t num);
```

```
char *strncat(char *destination, const char *source, size_t num);
char *strcpy(char *destination, const char *source);
char *strcat(char *destination, const char *source);
int strncmp(const char *str1, const char *str2, size_t num);
int strcmp(const char *str1, const char *str2);
char *strchr (char *str, int character);
void *memmove(void *destination, const void *source, size_t num);
void *memcpy(void *destination, const void *source, size_t num);
char *inttostr(size_t val, size_t base, char *destination);
bool isLetter(const char);
size_t copyBytes(byte *destination, const byte *source, size_t num);
size_t strcspn(const char *str1, const char *str2);
char *strtok( char *str, const char *delimiters );
size_t strspn(const char *str1, const char *str2);
char *trimwhitespace(char *str);
#ifdef __cplusplus
 }
#endif // __cplusplus

#endif // _STRING_H
```

Αρχείο: sysCallLib.c

```
#include "sysCallLib.h"
#include "registers.h"

makeSysCall2(void, ScPrintChar, char, charToPrint, byte, color)

makeSysCall1(int, freePhysicalPages, void*, ptr)

makeSysCall4(void*, memAlloc, size_t, length, dword, base_address, dword, mode, dword,
protection)
```

makeSysCall2(dword, createProcces, const char *, exepath, const char *, params)

makeSysCall1(void, exitProc, int, ret)

makeSysCall1(int, waitProc, int*, status)

makeSysCall3(dword, readFile, dword, fileHndlr, size_t, count, byte*, buffer)

makeSysCall4(dword, writeFile, dword, fileHndlr, size_t, count, byte*, buffer, int, flags)

makeSysCall2(dword, openFile, const char *, path, const char *, rw)

makeSysCall3(dword, seekFile, dword, fileHndlr, long int, offset, int, origin)

makeSysCall1(void, closeFile, dword, filehndlr)

makeSysCall1(dword, getMessage, proccesMsg *, msg)

makeSysCall1(dword, setProcProperties, dword, procDescr)

makeSysCall1(dword, setActiveProc, dword, procHndl)

makeSysCall4(void, draw, byte*, buffer, size_t, offset, size_t, count, dword, position)

makeSysCall3(dword, fileExists, const char *, path, dword *, attr, size_t *, fsize)

makeSysCall4(dword, getFileAttr, const char *, path, const dword, fromFile, size_t, countFiles, fileAttr *, fileAtt)

makeSysCall1(dword, createFile, const char *, path)

makeSysCall1(dword, loadDriver, const char *, driverpath)

Αρχείο: sysCallLib.h

```
#ifndef _SYSCALLLIB_H
#define _SYSCALLLIB_H

#include "type.h"

#ifdef __cplusplus
 extern "C"{
#endif // __cplusplus

typedef struct SproccesMsg{
   dword msgType;
```

```
   dword msg;
   dword param;
} proccesMsg;

typedef struct TfileAttr{
   char fName[128];
} fileAttr;

#define GET_FILE_ATTR_ERROR   ((size_t)-1)



#define PAGE_FRAME_SIZE     4096

#define     ALLOC_MODE_FIXED_ADD    0x01
#define     ALLOC_MODE_ANY_ADD      0x02

void ScPrintChar(char charToPrint, byte color);
int freePhysicalPages(void* );
void* memAlloc(size_t length, dword base_address, dword mode, dword protection);
dword createProcces(const char *exepath, const char *params);
void exitProc(int );
int waitProc(int*);
dword readFile(dword filehndlr, size_t count, byte* buffer);
dword writeFile(dword fileHndlr, size_t count, byte* buffer, int flags);
dword openFile(const char *path, const char *rw);
dword seekFile(dword filehndlr, long int offset, int origin);
void closeFile(dword filehndlr);
dword getMessage(proccesMsg *msg);
dword setProcProperties(dword procDescr);
dword setActiveProc(dword procHndl);
void draw(byte*, size_t, size_t, dword);
dword fileExists(const char*path, dword *attr, size_t *fsize);
```

```
dword getFileAttr(const char* fpath, const dword fromFile, size_t countFiles,  fileAttr *fileAtt);
dword createFile(const char *path);
dword loadDriver(const char *driverpath);

#ifdef __cplusplus
 }
#endif // __cplusplus

#define ScPrintChar_NR              0
#define freePhysicalPages_NR        1
#define memAlloc_NR                 2
#define createProcces_NR            3
#define exitProc_NR                 4
#define waitProc_NR                 5
#define readFile_NR                 6
#define writeFile_NR                7
#define openFile_NR                 8
#define seekFile_NR                 9
#define closeFile_NR                10
#define getMessage_NR               11
#define setProcProperties_NR        12
#define setActiveProc_NR            13
#define draw_NR                     14
#define fileExists_NR               15
#define getFileAttr_NR              16
#define createFile_NR               17
#define loadDriver_NR               18

/*From https://www.win.tue.nl/~aeb/linux/lk/lk-4.html*/
#define makeSysCall0(type, name)          \
type name(void){                          \
   size_t res;                            \
```

```
   __asm volatile("int $255;"              \
            :"=a" (res)             \
            :"0" (name##_NR)         \
            :);                    \
   return (type) res;                  \
}


#define makeSysCall1(type, name, type0, arg0)         \
type name(type0 arg0){                          \
   size_t res;                             \
   __asm volatile("int $255;"                    \
            :"=a" (res)                   \
            :"0" (name##_NR), "b" (arg0)         \
            :);                           \
   return (type) res;                        \
}

#define makeSysCall2(type, name, type0, arg0, type1, arg1)     \
type name(type0 arg0, type1 arg1){                     \
   size_t res;                                \
   __asm volatile("int $255;"                        \
            :"=a" (res)                      \
            :"0" (name##_NR), "b" (arg0), "c" (arg1)      \
            :);                              \
   return (type) res;                              \
}

#define makeSysCall3(type, name, type0, arg0, type1, arg1, type2, arg2)        \
type name(type0 arg0, type1 arg1, type2 arg2){                         \
   size_t res;                                          \
   __asm volatile("int $255;"                                  \
```

```
        :"=a" (res)                                     \
        :"0" (name##_NR), "b" (arg0), "c" (arg1), "d"(arg2)         \
        :);                                             \
   return (type) res;                                   \
}

#define makeSysCall4(type, name, type0, arg0, type1, arg1, type2, arg2, type3, arg3)       \
type name(type0 arg0, type1 arg1, type2 arg2, type3 arg3){                          \
   size_t res;                                                  \
   __asm volatile("int $255;"                                           \
        :"=a" (res)                                             \
        :"0" (name##_NR), "b" (arg0), "c" (arg1), "d"(arg2), "S"(arg3)          \
        :);                                                     \
   return (type) res;                                               \
}

#endif // _sysCallLib_H
```

Αρχείο: sysCallStructs.h

```
#ifndef _PROCCESS_STRUCT_H
#define _PROCCESS_STRUCT_H

#include "type.h"

typedef struct SproccesMsg{
   dword msgType, msg, param;
} proccesMsg;

#endif // _PROCCESS_STRUCT_H
```

Αρχείο: terminal.cc

```
#include "terminal.hpp"
#include "fileRead.hpp"
#include "fileWrite.hpp"
#include "sysCallLib.h"
#include "stdlib.h"

#define  SCREEN_WIDTH   80
#define  SCREEN_HEIGHT  25
#define  SCREEN_SIZE    ( (SCREEN_WIDTH)  *  (SCREEN_HEIGHT) )

/**??class termninalData. TerminnalRead{terminalData tD} Cterminal{setProcProperties;
setStdRead = terminalData} **/
```

```
static termninalData tD;  /**?? support constructor at global*/

Cterminal::Cterminal()
{
   setProcProperties(1);
   tD.init();
   setStdRead(&TerminlalRead); //sdtRead = &read;
   setStdWrite(&TerminlalWrite);
}

bool termninalData::init(void)
{
   rowStart = 0;
   cursor = 0;
   screen = (byte *) malloc(SCREEN_SIZE);
   stdBuff.init(2048 * 16);

   if(screen == NULL)
   {
      while(1);
      return false;
   }

   return true;
}

dword TerminlalWrite(size_t count, byte* buffer)
{
   size_t i, j, k = 0;

   for(i = 0;  i < count;  i++)
```

```
{
   if(tD.stdBuff.insert(buffer[i]) )
      tD.rowStart--;

   if(tD.rowStart < 0)
      tD.nextRow();

   if(tD.incCursor(buffer[i] == '\n'))
      tD.nextRow();
}

j = tD.rowStart;

while(k < SCREEN_SIZE && j < tD.stdBuff.getSize())
{
   if(tD.stdBuff[j] == '\n' )
   {
      tD.screen[k++] = ' ';
      while(k % SCREEN_WIDTH)tD.screen[k++] = ' ';
   }
   else
      tD.screen[k++] = tD.stdBuff[j];

   j++;
}

while(k < SCREEN_SIZE)
   tD.screen[k++] = ' ';

draw(tD.screen, 0, SCREEN_SIZE,  tD.cursor);

return i;
```

```
}

dword TerminlalRead(size_t count, byte* buffer)
{
   proccesMsg msg;
   char ch;
   size_t i = 0, k, j;


   do
   {
      ch = '\0';
      getMessage(&msg);
      j = 0xFFFFFF;

      if(msg.msgType == 0x1)
      {
         k = 0;
         ch = msg.msg;


         if(msg.param == 73)  //PageUp
            j = tD.goNLinesBack(SCREEN_HEIGHT);
         else if(msg.param == 81)  //PageUp
            j = tD.goNLinesFront(SCREEN_HEIGHT);
         else if(msg.param == 72)  //ArrowUp
            j = tD.goNLinesBack(1);
         else if(msg.param == 80)  //ArrowDown
            j = tD.goNLinesFront(1);
         else if(msg.msg == 8)  //VK_BACKSPACE
         {
            if(i > 0)
```

```
    {
        buffer[i--] = '\0';
        tD.stdBuff.removeLast();

        if(tD.decCursor())
            tD.goNLinesBack(1);

    }
        j = tD.rowStart;
}
else if(msg.msg != 0)
{
    if(tD.stdBuff.insert(ch) )
        tD.rowStart--;

    if(tD.rowStart < 0)
        tD.nextRow();

    buffer[i++] = ch;

    if(tD.incCursor(ch == '\n'))
        tD.nextRow();

    j = tD.rowStart;
}

if(j != 0xFFFFFF)
{

    while(k < SCREEN_SIZE && j < tD.stdBuff.getSize())
    {
        if(tD.stdBuff[j] == '\n' )
```

```
            {
               tD.screen[k++] = ' ';
               while(k % SCREEN_WIDTH)tD.screen[k++] = ' ';
            }
            else
               tD.screen[k++] = tD.stdBuff[j];

            j++;
         }

         while(k < SCREEN_SIZE)
            tD.screen[k++] = ' ';

         draw(tD.screen, 0, SCREEN_SIZE,  tD.cursor);
      }
   }

}while( ((msg.msg != '\n' ) || (msg.msgType != 0x1)) && i < count );

if(i > 0)
   buffer[i-1] = '\0';

return i;
}

bool termninalData::incCursor(bool newLine)
{
   size_t row = cursor >> 0x10;

   if( (cursor & 0xFFFF) < SCREEN_WIDTH - 1  && !newLine)
   {
      cursor++;
```

```
        return false;
    }

    if(row < SCREEN_HEIGHT - 1)
    {
        cursor = (((cursor >> 0x10) + 1) << 0x10);
        return false;
    }

    cursor &= 0xFFFF0000;
    return true;
}

bool termninalData::decCursor(void)
{
    size_t row = cursor >> 0x10;

    if( (cursor & 0xFFFF) > 0)
    {
        cursor--;
        return false;
    }

    if(row > 0)
    {
        cursor = ((row - 1) << 0x10) | ((SCREEN_WIDTH - 1) & 0xFFFF);
        return false;
    }

    cursor = (SCREEN_WIDTH - 1) & 0xFFFF;
    return true;
}
```

```
size_t termninalData::goNLinesBack(size_t linesBack)
{
   size_t j = 0;

   while(rowStart > 0 && j < linesBack)
   {
      goToPrevLine();
      j++;
   }

   return rowStart;
}

void termninalData::goToPrevLine(void)
{
   size_t i = 0;

   if(rowStart == 0)
      return;



   do
   {
      rowStart--;
      i++;
   }while(rowStart > 0  && stdBuff[rowStart - 1] != '\n' && i < SCREEN_WIDTH);
}

size_t termninalData::goNLinesFront(size_t linesFront)
{
```

```
    size_t lastLine = stdBuff.getSize(), i = 0;

    while(lastLine > 0  && stdBuff[lastLine-1] != '\n'  && i < SCREEN_WIDTH )
    {
        lastLine--;
        i++;
    }

    i = 0;
    while(rowStart < lastLine - 1 && i < linesFront)
    {
        goToNextLine();
        i++;
    }

    return rowStart;
}

void termninalData::goToNextLine(void)
{
    size_t i = 0;
    size_t lastLine = stdBuff.getSize();

    if(rowStart >= lastLine)
        return;


    do
    {
        rowStart++;
        i++;
    }while(rowStart < lastLine  && stdBuff[rowStart - 1] != '\n' && i < SCREEN_WIDTH);
```

```
}

void termninalData::nextRow(void)
{
   size_t j = rowStart, col = 0, bfSz = stdBuff.getSize();

   while(stdBuff[j] != '\n' && col < SCREEN_WIDTH -1 && j < bfSz)
   {
      j++;
      col++;
   }

   rowStart = j+1;
}
```

Αρχείο: terminal.hpp

```
#ifndef _TERMINAL_H
#define _TERMINAL_H

#include "dataStructures/circularBuffer.hpp"

class termninalData{
   public:
      Ccirc_buffer stdBuff;
            size_t rowStart;
            byte *screen;
            dword cursor;
            bool incCursor(bool newLine);
```

```
        bool decCursor(void);
        bool init(void);
        void nextRow(void);
        size_t goNLinesBack(size_t linesBack);
        void goToPrevLine(void);
        size_t goNLinesFront(size_t linesBack);
        void goToNextLine(void);
};

class Cterminal{
        public:
    Cterminal();
    //dword read(size_t count, byte* buffer);
};

dword TerminlalRead(size_t count, byte* buffer);
dword TerminlalWrite(size_t count, byte* buffer);
#endif // _TERMINAL_H
```

Αρχείο: type.h

```
#ifndef _TYPE_H
#define _TYPE_H


#ifdef __cplusplus
#define NULL 0
#else
#define NULL ((void*)0)
#endif

typedef char    s8bit;
```

```
typedef unsigned char   u8bit;
typedef unsigned char   byte;

#ifndef __cplusplus
typedef enum {
  false = 0,
  true = 1
} bool;
#endif // __cplusplus

typedef short int   s16bit;
typedef unsigned short int  u16bit;
typedef unsigned short int  word;

#ifdef __x86_64__
#error 64bit code is not supported!!
#else
#define __WORDSIZE 32
#endif // __x86_64__

#if __WORDSIZE == 32
typedef int s32bit;
typedef unsigned int u32bit;
typedef unsigned int dword;
typedef unsigned int size_t;
#endif // _WORDSIZE

typedef long long qword;


typedef dword   register32;
typedef word    register16;
```

```
typedef register32 address;

typedef int int32_t;
typedef unsigned int uint32_t;

typedef long long int64_t;
typedef unsigned long long uint64_t;

#endif /// _TYPE_H
```

Αρχείο: circularBuffer.cc

```
#include "circularBuffer.hpp"
#include "../sysCallLib.h"
#include "../stdlib.h"


bool Ccirc_buffer::init(size_t sz)
{
   buffer = (char*) malloc(sz);
   if(buffer == NULL)
   {
      maxSize = currSize = bstart = bend = 0;
```

```
      return true;
   }

   maxSize = sz;
   currSize = bstart = 0;
   bend = 0;
   return false;
}

char& Ccirc_buffer::operator[] (size_t indx) const
{
   return buffer[(indx + bstart) % currSize];
}


bool Ccirc_buffer::insert(const char ch)
{
   bool startInc = false;

   buffer[bend] = ch;
   bend = (bend == maxSize - 1) ? 0 : bend + 1;

   if(bend == bstart)
   {
      bstart = (bstart == maxSize - 1) ? 0 : bstart + 1;
      startInc = true;
   }

   if(!startInc)
      currSize++;

   return startInc;
```

```
}

void Ccirc_buffer::removeLast(void)
{

   if(!currSize)
      return;

   bend = (!bend) ? maxSize-1 : bend - 1;
   currSize--;

}

size_t Ccirc_buffer::getSize(void) const
{
   return currSize;
}
```

Αρχείο: circularBuffer.hpp

```
#ifndef CIRCULAR_BUFFER_H
#define CIRCULAR_BUFFER_H

#include "../type.h"

class Ccirc_buffer{
   private:
      size_t bstart;
      size_t bend;
      size_t maxSize;
      size_t currSize;
      char *buffer;
```

```
    void incIndex(size_t indx);
        public:
    bool init(size_t);
    char& operator[] (size_t indx) const;
    bool insert(const char ch);
    void removeLast(void);
    //void insertAtEnd(const char ch);
    //void removeFromEnd(void);
    //void moveGap(const size_t newGapStart);
    //size_t getGapIndx(void) const;
    //size_t getEoGapPos(void) const;
    size_t getSize(void) const;

};



#endif // CIRCULAR_BUFFER_H
```

Αρχείο: gapBuffer.cc

```
#include "gapBuffer.hpp"
#include "../sysCallLib.h"
#include "../stdlib.h"

Cgap_buffer::Cgap_buffer(void)
{
   gBuffer = NULL;
   bufferSz = 0;
   gap_start = gap_end = 0;
}
```

```
void Cgap_buffer::assignBuffer(char *buffer, size_t sz, size_t gapStart, size_t gapEnd)
{
   gBuffer = buffer;
   bufferSz = sz;
   gap_start = gapStart;
   gap_end = gapEnd;
}


char& Cgap_buffer::operator[] (size_t indx) const
{
   if(indx >= gap_start)
      indx += gap_end - gap_start;

   return gBuffer[indx];
}


void Cgap_buffer::insert(const char ch)
{
   if(gap_start  >= gap_end)
      return;

   gBuffer[gap_start++] = ch;
}

void Cgap_buffer::insertAtEnd(const char ch)
{
   if(gap_start  >= gap_end)
      return;
```

```
   gBuffer[--gap_end] = ch;
}

void Cgap_buffer::removeFromEnd(void)
{
   if(gap_end >= bufferSz)
      return;

   gap_end++;
}

void Cgap_buffer::moveGap(const size_t newGapStart)
{
   if(newGapStart + (gap_end - gap_start) > bufferSz)
      return;

   while(gap_start < newGapStart)
      gBuffer[gap_start++] = gBuffer[gap_end++];


   while(gap_start > newGapStart)
      gBuffer[--gap_end] = gBuffer[--gap_start];

}

size_t Cgap_buffer::getGapIndx(void) const
{
   return gap_start;
}

size_t Cgap_buffer::getEoGapPos(void) const
{
```

```
   return gap_end;
}

size_t Cgap_buffer::getSize(void) const
{
   return gap_start + (bufferSz - gap_end);
}
```

Αρχείο: gapBuffer.hpp

```
#ifndef GAP_BUFFER_H
#define GAP_BUFFER_H

#include "../type.h"

class Cgap_buffer{
   private:
      size_t gap_start;
      size_t gap_end;
      size_t bufferSz;
      char *gBuffer;
          public:
      Cgap_buffer(void);
      void assignBuffer(char *buffer, size_t sz, size_t gapStart, size_t gapEnd);
      char& operator[] (size_t indx) const;
      void insert(const char ch);
      void insertAtEnd(const char ch);
      void removeFromEnd(void);
      void moveGap(const size_t newGapStart);
      size_t getGapIndx(void) const;
      size_t getEoGapPos(void) const;  ///??private member
      size_t getSize(void) const;
```

```
};



#endif // GAP_BUFFER_H
```

CMD

Αρχείο: cd.cc

```
#include "cmd.hpp"
#include "../Os_cLibrary/stdio.h"
#include "../Os_cLibrary/sysCallLib.h"
#include "../Os_cLibrary/string.h"
#include "../Os_cLibrary/osUtils.h"


void cmd_session::cd(char *params)
{
   char tmp[MAX_PATH_LEN];
```

```
   dword attr;

   strcpy(tmp, dir);

   if( !pathIsAbsolute(params) )
      pathToAbs(tmp, params);
   else
      strcpy(tmp, params);


   removeGoToParentFromPath(tmp+3);

   if( fileExists(tmp, &attr, (dword*) NULL) && attr == 0)
      strcpy(dir, tmp);
   else
      printf("Not a valid path\n");
}
```

Αρχείο: cls.cc

```
#include "cmd.hpp"
#include "../Os_cLibrary/stdio.h"
#include "../Os_cLibrary/sysCallLib.h"

byte emptyDraw[80*25];

void cmd_session::cls(char *params)
{
   memset(emptyDraw, ' ', 80*25);
   draw(emptyDraw, 0, 80*25, 0);
}
```

Αρχείο: cmd.cc

```
#include "cmd.hpp"
#include "../Os_cLibrary/stdio.h"
#include "../Os_cLibrary/sysCallLib.h"
#include "../Os_cLibrary/string.h"

cmd_session::cmd_session()
{
   strcpy(dir, "A:/");
}

bool cmd_session::nextCommand()
{
   char *in;
```

```
   printf(dir);
   printf(">");
   in = trimwhitespace(gets(input) );

   strncpy(command, in, strcspn(in, " \t\n\r") );
   command[strcspn(in, " \t\n\r")] = '\0';

   strcpy(params, trimwhitespace(in + strcspn(in, " \t\n\r")) );

///?? command, params not as function params
   if( !strcmp(command, "echo" ) )
      echo(params);
   else if( !strcmp(command, "cls" ) )
      cls(params);
   else if( !strcmp(command, "cd" ) )
      cd(params);
   else if( !strcmp(command, "ls" ) )
      ls(params);
   else if( !strcmp(command, "drvload" ) )
      loadDriver(params);
   else if( !strcmp(command, "fcreate" ) )
      createFile(params);
   else if( execute(command, params) )
      printf("Not such command!...\n");

   /*
   if( !strcmp(session.lastCommand(), "cd" ) )
      ///cd
   else if( !strcmp(session.lastCommand(), "..." ) )
      ///...
   else
```

```
      ///default
   */

   return true;
}
```

Αρχείο: cmd.hpp

```
#ifndef CMD_H
#define CMD_H

#include "../Os_cLibrary/osUtils.h"

class cmd_session{
   private:
      char dir[MAX_PATH_LEN];
      char input[MAX_PATH_LEN];
      char command[MAX_PATH_LEN];
      char params[256];
            void echo(char *params);
            void cls(char *params);
            void cd(char *params);
            void ls(char *params);
      int execute(const char *command, const char *params);
          public:
      cmd_session();
            bool nextCommand();
};

#endif // CMD_H
```

Αρχείο: echo.cc

```
#include "cmd.hpp"
#include "../Os_cLibrary/stdio.h"


void cmd_session::echo(char *params)
{
```

```
   printf(params);
   printf("\n");
}
```

Αρχείο: execute.cc

```
#include "cmd.hpp"
#include "../Os_cLibrary/sysCallLib.h"
#include "../Os_cLibrary/stdio.h"
#include "../Os_cLibrary/osUtils.h"
#include "../Os_cLibrary/string.h"

int cmd_session::execute(const char *command, const char *params)
```

```
{
   char tmp[MAX_PATH_LEN];
   dword attr;
   if( !strlen(command) )
      return 0;

   normalizePaths(tmp, command, dir);
   if( fileExists(tmp, &attr, (dword *) NULL) )
   {
      if(attr == 1)
      {
         createProcces(tmp, params);
         return 0;
      }
   }

   return 1;
}
```

Αρχείο: ls.cc

```
#include "cmd.hpp"
#include "../Os_cLibrary/stdio.h"
#include "../Os_cLibrary/string.h"
#include "../Os_cLibrary/sysCallLib.h"

void cmd_session::ls(char *params)
{
   size_t i = 0, cnt;
   fileAttr fileAtt;
   char tmp[MAX_PATH_LEN];
```

```
  strcpy(tmp, dir);

  if(!strlen(params))
    strcpy(params, dir);

  if( !pathIsAbsolute(params) )
    pathToAbs(tmp, params);
  else
    strcpy(tmp, params);

  removeGoToParentFromPath(tmp+3);

  cnt = getFileAttr(tmp, i, 1, &fileAtt);
  while( cnt > 0  && GET_FILE_ATTR_ERROR != cnt )
  {
    fileAtt.fName[127] = '\0';
    printf(fileAtt.fName );
    printf("\n");
    i++;
    cnt = getFileAttr(tmp, i, 1, &fileAtt);
  }

  if(GET_FILE_ATTR_ERROR == cnt)
    printf("Path is invalid\n");


}
```

Αρχείο: main.cc

```
#include "cmd.hpp"
#include "../Os_cLibrary/stdio.h"
#include "../Os_cLibrary/sysCallLib.h"
#include "../Os_cLibrary/stdlib.h"
#include "../Os_cLibrary/terminal.hpp"

void _main(void) __asm ("__main");

void _main(void)
{
        cmd_session session;
```

```
   //setProcProperties(1);
   stdInit();
   Cterminal terminal;
   ///setStdInput(HCONSOLE);
   ///setStdOutput(HCONSOLE);


   while( session.nextCommand() )
   {

   }

   //return 0xbeff;
}
```

NOTEPAD

Αρχείο: main.cc

```
#include "../Os_cLibrary/stdio.h"
#include "../Os_cLibrary/sysCallLib.h"
#include "../Os_cLibrary/stdlib.h"
#include "../Os_cLibrary/dataStructures/gapBuffer.hpp"

#define  SCREEN_WIDTH   80
#define  SCREEN_HEIGHT  25
#define  SCREEN_SIZE    ( (SCREEN_WIDTH)  *  (SCREEN_HEIGHT) )

#define  GAP_BUFFER_SZ   512   //1048576   //1M
```

```
typedef struct TcursorPos{
   word column;
   word line;
} cursorPos;

size_t gColumn;  //?? = 10

void _main(char params[]) __asm ("__main");



static void refreshScreen(Cgap_buffer *gapBuffer, size_t loffset, const size_t textSz, byte
*screen, size_t colOffs, cursorPos screenXY)
{
   size_t screenCur = 0, textCurr = loffset, i, rowNum;



   for(rowNum = 0;  rowNum < SCREEN_HEIGHT;  rowNum++)
   {

      i = 0;
      while(textCurr < textSz  &&  (*gapBuffer)[textCurr] != '\n'   &&  i < colOffs)
      {
         textCurr++;
         i++;
      }

      i = 0;
      while(i < SCREEN_WIDTH  &&  (*gapBuffer)[textCurr] != '\n'  &&  textCurr < textSz)
      {
         screen[screenCur++] = (*gapBuffer)[textCurr++];
         i++;
      }
```

```
    if(i < SCREEN_WIDTH)
      while(i++ <  SCREEN_WIDTH)
        screen[screenCur++] = ' ';
    else
      while( (*gapBuffer)[textCurr] != '\n' && textCurr < textSz)
        textCurr++;

    if((*gapBuffer)[textCurr]  == '\n' && textCurr < textSz)
      textCurr++;
//if(textCurr>= textSz){printf("OKOKOK  %x    %x   %x\n", textCurr, textSz, loffset); while(1);}
  }


  draw(screen, 0, SCREEN_SIZE,  ( ((dword) screenXY.line) << 16) | screenXY.column ) ;
}

static void bufferNextLine(const Cgap_buffer *gapBuffer, const size_t textSz, cursorPos
*screenXY, size_t *scrnLoff)
{
  if(screenXY->line < SCREEN_HEIGHT - 1)
    screenXY->line++;
  else
  {
    while((*gapBuffer)[(*scrnLoff)] != '\n' && (*scrnLoff) < textSz)
      (*scrnLoff)++;

    if((*scrnLoff) < textSz)
      (*scrnLoff)++;
  }
}
```

```
static void bufferPrevLine(const Cgap_buffer *gapBuffer, cursorPos *screenXY, size_t
*scrnLoff)
{
   if(screenXY->line > 0)
      screenXY->line--;
   else
   {
      if((*scrnLoff) > 0)
         (*scrnLoff)--;

      while( (*scrnLoff) > 0  && (*gapBuffer)[(*scrnLoff) - 1] != '\n')
         (*scrnLoff)--;

   }
}

static void moveNextCol(Cgap_buffer *gapBuffer, const size_t textSz, size_t *textCurr,
cursorPos *screenXY, size_t *scrnCoff, size_t *scrnLoff )
{
   size_t pos = gapBuffer->getGapIndx();

   if(  pos >= textSz)  ///Eof reached
      return;
//printf("345dfgrdfgdfg\n,"); while(1);

   if((*gapBuffer)[pos] == '\n')
   {
      bufferNextLine(gapBuffer, textSz, screenXY, scrnLoff);

      screenXY->column = 0;
      (*scrnCoff) = 0;
   }
```

```
   else if(screenXY->column < SCREEN_WIDTH - 1)
      screenXY->column++;
   else
      (*scrnCoff)++;

   gColumn = (*scrnCoff) + screenXY->column;
   gapBuffer->moveGap(pos + 1); //(*textCurr)++;
}

static void movePrevCol(Cgap_buffer *gapBuffer, const size_t textSz, size_t *textCurr,
cursorPos *screenXY, size_t *scrnCoff, size_t *scrnLoff )
{
   size_t lnLength, pos = gapBuffer->getGapIndx();
   int c;

   if( ! pos )
      return;

   gapBuffer->moveGap(--pos);

   if((*gapBuffer)[pos] == '\n')
   {
      lnLength = 0;
      c = (int) pos - 1;
      while( c >= 0  &&  (*gapBuffer)[c] != '\n'  && lnLength < SCREEN_WIDTH - 1 )
      {
         lnLength++;
         c--;
      }

      (*scrnCoff) = 0;
      while(c >= 0  && (*gapBuffer)[c] != '\n')
      {
```

```
            (*scrnCoff)++;
            c--;
        }

        bufferPrevLine(gapBuffer, screenXY, scrnLoff);
        screenXY->column = lnLength;
    }
    else if(screenXY->column > 0)
        screenXY->column--;
    else
        (*scrnCoff)--;

    gColumn = (*scrnCoff) + screenXY->column;
}

int fmin(int a, int b)
{
    return  a < b ? a : b;
}

int fmax(int a, int b)
{
    return  a > b ? a : b;
}

static void moveNextLine(Cgap_buffer *gapBuffer, const size_t textSz, size_t *textCurr,
cursorPos *screenXY, size_t *scrnCoff, size_t *scrnLoff )
{
    size_t newLnCol, oldScreenCol, curr = gapBuffer->getGapIndx(), lineSz = 0, prevLnCol =
screenXY->column + *scrnCoff;

    if(curr >= textSz)
        return;
```

```
  while(curr < textSz  &&  (*gapBuffer)[curr] !=  '\n')
    curr++;

  if((*gapBuffer)[curr]  != '\n')
    return;

  gapBuffer->moveGap(++curr);
  bufferNextLine(gapBuffer, textSz, screenXY, scrnLoff);

  while(curr < textSz  &&  (*gapBuffer)[curr] !=  '\n')
  {
    curr++;
    lineSz++;
  }

  newLnCol = fmin((int) lineSz, (int) gColumn);
  gapBuffer->moveGap( gapBuffer->getGapIndx() + newLnCol);
  oldScreenCol = screenXY->column;

  if(newLnCol > prevLnCol)
  {
    screenXY->column = fmin((int) oldScreenCol + (newLnCol - prevLnCol), (int)
SCREEN_WIDTH - 1);

    if(screenXY->column + (newLnCol - prevLnCol) > SCREEN_WIDTH - 1)
      (*scrnCoff) += (newLnCol - prevLnCol) - (screenXY->column - oldScreenCol); //(*scrnCoff)
+= (screenXY->column + (newLnCol - prevLnCol)) - (SCREEN_WIDTH - 1);
  }
  else if(newLnCol < prevLnCol)
  {
    screenXY->column = fmax( (int) oldScreenCol - (int) (prevLnCol - newLnCol), 0);
```

```
      if((int) screenXY->column - (int) (prevLnCol - newLnCol) < 0)
         (*scrnCoff) -= ( (prevLnCol - newLnCol) - (oldScreenCol - screenXY->column) );
   }

}

static void movePrevLine(Cgap_buffer *gapBuffer, const size_t textSz, size_t *textCurr,
cursorPos *screenXY, size_t *scrnCoff, size_t *scrnLoff )
{
   size_t newLnCol, oldScreenCol, curr = gapBuffer->getGapIndx(), lineSz = 0, prevLnCol =
screenXY->column + *scrnCoff;

   if( !curr )
      return;

   curr--;
   while(curr > 0 &&  (*gapBuffer)[curr] !=  '\n')
      curr--;

   if((*gapBuffer)[curr]  != '\n')
      return;

   gapBuffer->moveGap(curr);
   bufferPrevLine(gapBuffer, screenXY, scrnLoff);

   while(curr > 0  &&  (*gapBuffer)[curr - 1] !=  '\n')
   {
      curr--;
      lineSz++;
   }

   newLnCol = fmin((int) lineSz, (int) gColumn);
```

```
   gapBuffer->moveGap(gapBuffer->getGapIndx() - (lineSz - newLnCol) ); //(*textCurr) -=  lineSz
- newLnCol;
   oldScreenCol = screenXY->column;


   if(newLnCol > prevLnCol)
   {
      screenXY->column = fmin((int) oldScreenCol + (newLnCol - prevLnCol), (int)
SCREEN_WIDTH - 1);


      if(screenXY->column + (newLnCol - prevLnCol) > SCREEN_WIDTH - 1)
         (*scrnCoff) += (newLnCol - prevLnCol) - (screenXY->column - oldScreenCol);
   }
   else if(newLnCol < prevLnCol)
   {
      screenXY->column = fmax( (int) oldScreenCol - (int) (prevLnCol - newLnCol), 0);


      if((int) screenXY->column - (int) (prevLnCol - newLnCol) < 0)
         (*scrnCoff) -= ( (prevLnCol - newLnCol) - (oldScreenCol - screenXY->column) );
   }


}


static void moveEndOfLine(Cgap_buffer *gapBuffer, const size_t textSz, size_t *textCurr,
cursorPos *screenXY, size_t *scrnCoff )
{
   size_t oldColumn = screenXY->column, pos = gapBuffer->getGapIndx();
   size_t oldTextCurr = pos;


   if(pos >= textSz)
      return;


   while( (*gapBuffer)[pos] != '\n' && pos < textSz)
      gapBuffer->moveGap(++pos);
```

```
   screenXY->column = fmin((int) oldColumn + (pos - oldTextCurr), (int) SCREEN_WIDTH - 1);
   *scrnCoff += (pos - oldTextCurr) - (screenXY->column - oldColumn);
   gColumn = (*scrnCoff) + screenXY->column;
}

static bool isSpace(char ch)
{
   return ch == ' ';
}

static void moveBeginOfLine(Cgap_buffer *gapBuffer, const size_t textSz, size_t *textCurr,
cursorPos *screenXY, size_t *scrnCoff )
{
   size_t pos = gapBuffer->getGapIndx(), oldTextCurr = pos, oldColumn = screenXY->column;
   size_t beginOfText, beginOfLine = pos;

   if(!textSz)
      return;

   if( pos >= textSz  && (*gapBuffer)[textSz-1] == '\n' )
      return;

   while( beginOfLine  && (*gapBuffer)[beginOfLine - 1] != '\n' )
      beginOfLine--;

   beginOfText = beginOfLine;

   while(beginOfText < textSz  && isSpace((*gapBuffer)[beginOfText]) &&
(*gapBuffer)[beginOfText] != '\n')
      beginOfText++;
```

```
   if(beginOfText > beginOfLine)
      beginOfText--;

   if(pos == beginOfText)
   {

      screenXY->column = 0;
      *scrnCoff = 0;
      gapBuffer->moveGap(beginOfLine);
   }
   else
   {
      screenXY->column = fmin( fmax((int) oldColumn + (beginOfText - pos), 0), (int)
SCREEN_WIDTH - 1);
      *scrnCoff +=   (beginOfText - pos) - (screenXY->column - oldColumn);
      gapBuffer->moveGap(beginOfText);
   }


   gColumn = (*scrnCoff) + screenXY->column;
}

void _main(char params[])
{
   dword hfile;
   size_t fsize, textCurr = 0, colOffset = 0, loffset = 0, bytesRead;
   char *text;
   byte *screen;
   cursorPos screenXY = {0, 0};
   proccesMsg msg;
   Cgap_buffer gapBuffer;
   bool quit = false;
```

```
gColumn = 0;



setProcProperties(1);
stdInit();

//printf(params);
/*screen = (byte *) malloc(SCREEN_SIZE);
sprintf((char*) screen, params );
for(int i = 0;  i < 80;  i++)
   if(screen[i] == '\0') screen[i] = '!';
draw(screen, 0, 80, 0);
getMessage(&msg);
getMessage(&msg);
exitProc(0x01);*/

if( !fileExists(params, (dword*) NULL, &fsize) )
{
   draw((byte*)"Error", 0, 5, 0);
   //printf("Error:: Notepad: %s file does not exits\n", params);
   while(1);
   exitProc(0x1);
}

text = (char *) malloc(fsize + GAP_BUFFER_SZ);
if( text == NULL )
{
   printf("Error:: Notepad: Failed to allocate 0x%x bytes.\n", fsize + GAP_BUFFER_SZ );
   while(1);
}
```

```
screen = (byte *) malloc(SCREEN_SIZE);
if( screen == NULL )
{
   printf("Error:: Notepad: Failed to allocate 0x%x bytes for video output.\n", SCREEN_SIZE);
   while(1);
}

hfile = openFile(params, "");
if( hfile == (size_t) -1)
{
   printf("Error:: Notepad: Failed to open file %s\n", params);
   while(1);
}



readFile(hfile, fsize, (byte*) text + GAP_BUFFER_SZ);
gapBuffer.assignBuffer(text, fsize + GAP_BUFFER_SZ, 0, GAP_BUFFER_SZ  );

while(!quit)
{
   getMessage(&msg);

   if(msg.msgType == 0x1 )
   {
      switch (msg.param){
      case 01:  ///VK_ESC
         quit = true;
         break;
      case 66:  ///VK_F8
         gapBuffer.moveGap(gapBuffer.getSize() );
```

```
            writeFile(hfile, gapBuffer.getSize(), (byte*) text, 1);
            break;

        case 77:  ///VK_RIGHT
            moveNextCol(&gapBuffer, fsize, &textCurr, &screenXY, &colOffset, &loffset);
            break;

        case 75:  ///VK_LEFT
            movePrevCol(&gapBuffer, fsize, &textCurr, &screenXY, &colOffset, &loffset);
            break;

        case 80:  ///VK_DOWN
            moveNextLine(&gapBuffer, fsize, &textCurr, &screenXY, &colOffset, &loffset);
            break;

        case 72:  ///VK_UP
            movePrevLine(&gapBuffer, fsize, &textCurr, &screenXY, &colOffset, &loffset);
            break;

        case 71:  ///VK_HOME
            moveBeginOfLine(&gapBuffer, fsize, &textCurr, &screenXY, &colOffset);
            break;

        case 73:  ///VK_PAGE_UP
            printf("textCurr = %x, loffset = %x, colOffset = %x  \n", textCurr, loffset, colOffset);
            printf("line = %x, column = %x \n", screenXY.line,  screenXY.column);
            break;

        case 81:  ///VK_PAGE_DOWN
            printf("textCurr = %x, loffset = %x, colOffset = %x  \n", textCurr, loffset, colOffset);
            printf("line = %x, column = %x \n", screenXY.line,  screenXY.column);
            break;
```

```
        case 79:  ///VK_END
           moveEndOfLine(&gapBuffer, fsize, &textCurr, &screenXY, &colOffset);
           break;

        case 70:  ///SCROLL LOCK
           printf("File Size %x bytes\n", fsize);
           printf("StartGap = %x, StartEnd = %x, loffset = %x, colOffset = %x  \n",
gapBuffer.getGapIndx(), gapBuffer.getEoGapPos(), loffset, colOffset);
           printf("line = %x, column = %x \n", screenXY.line,  screenXY.column);
           break;

        default :
           if(msg.msg == 0x08)  ///VK_BACKSPACE
           {
              if( gapBuffer.getGapIndx() )
              {
                 movePrevCol(&gapBuffer, fsize, &textCurr, &screenXY, &colOffset, &loffset);
                 gapBuffer.removeFromEnd();
                 fsize = gapBuffer.getSize();
              }
           }
           else if(msg.msg == 0x7F) ///VK_DELETE
           {
              gapBuffer.removeFromEnd();
              fsize = gapBuffer.getSize();
           }
           else if(msg.msg)
           {
              gapBuffer.insertAtEnd(msg.msg);
              fsize = gapBuffer.getSize();
              moveNextCol(&gapBuffer, fsize, &textCurr, &screenXY, &colOffset, &loffset);
           }
```

```
        } ///switch


        if(msg.param != 70) refreshScreen(&gapBuffer, loffset, fsize, screen, colOffset,
screenXY);
      }


   }


   closeFile(hfile);


   exitProc(0x0);
}
```